CHAPTER 4

How to Build an Environment

Structures of Mathematical Subjects

Before proceeding with the building of Environments, it is important to understand that there are at least two ways of organizing — of structuring — a mathematical subject. The first is universal, and familiar to all students. It is logical structure. The second is hardly known, but has certain advantages over the first. It is called, in this book, problem-solving structure. I will try to make the difference clear, because the difference is important. However, let me emphasize at the start that the *content* of the subject is the same in both cases. I can organize the parts of a car in many different ways — by part number, or by name, or by function, or by location in the car — without destroying their relationships, e.g., that the pistons go in the cylinders in the engine block, and that they are attached to the crankshaft.

Logical Structure

Ever since Euclid's *Elements*, the great textbook on plane geometry, was first published in 323 b.c. (or, more correctly, first made available for copying, since there were no printing presses in those days), mathematical subjects (at least at the college level) have been presented in the form: initial definitions, axioms, definitions, theorem, proof, definitions, theorem, proof, ... , with implied rules for constructing proofs (the rules you learn in a course on mathematical logic). In other words, mathematical textbooks, and courses in mathematics, always present the *logical structure* of the subject. They assume that the only way you can solve a problem in chapter n is by having mastered chapters 1 through $n - 1$. This idea can be expressed in various ways: "If you want to tell the time, you need to know how the watch is built;" "If you want to go from point A to point B, you need to understand the territory first," or, in other words, "You have to learn everything in order to do anything."

This structure has dictated the form of learning mathematical subjects for centuries. It has been the structure of virtually all mathematical textbooks, and, indeed of all textbooks in the sciences. To a modern student, nothing seems more obvious than that you have to learn the subject, or concept, before you can apply it.

But that is by no means the only structure that a mathematical subject has. There is at least one other structure that I hope the following examples will make clear.

Suppose someone asks you, "How do I get from Berkeley to Palo Alto?" There are at least two ways you can reply: one is by handing the person a map of the San Francisco Bay Area and saying, simply, "Here, follow the map." Another way is to give the person explicit directions: "Get on Highway 80 going south, veer right to the Bay Bridge, go across the Bay Bridge and then get on Highway 101 and follow it for about 30 miles until you see an exit sign saying 'University Ave., Palo Alto'. Turn off at that exit, bear right, and keep following University Ave. to downtown Palo Alto".

In the first case, the person has to figure out his or her own route, and, if he or she doesn't know Bay Area traffic very well, he or she might choose a route that is much slower than another that an experienced commuter might recommend. The person, in effect, *has to study all of the territory in order to figure out how to make use of part of it.* His or her task is similar to that of a person who is attempting to learn to use a software system from a traditional manual. In the second case, the person benefits from the experience of the person who gives him or her explicit directions. Of course, if one of the roads happens to be closed, then the traveler will either have to proceed by trial-and-error, or by asking someone, or by getting a map. Which is why in a complete Environment, every set of tasks that can be performed on a given thing always includes the *optional* task, "Get background material on ..." — in other words, "Study the roadmap". The giant step forward, in the case of Environments, is that it is not *always* necessary to study the roadmap first.

One more example: suppose you have, on a sheet of paper, an arrangement of squares, triangles, and circles. This is the "logical structure" of the "subject" which consists of these squares, triangles, and circles. Now suppose there is another person, who is your student, and by dint of hard work, you manage to convey to him or her, by words alone, how the squares, triangles and circles are laid out on the paper. (You are not allowed to simply hand him the paper because that would be the equivalent of transferring part of your understanding of a real-life subject directly from your brain into a student's.) Eventually, you are convinced that he or she has managed to construct a close approximation of what is on your page.

Now once this has been done, you can indiscriminately ask the person anything you want about the subject. "How many triangles are below and to the right of a circle that is in the upper left quadrant of the page?" "Is there a triangle on the page such that, if you moved it less than two inches to the right, it would be directly under another triangle?" Any question you asked would be answerable — at the price of the student's having had to duplicate your entire page.

From here on in this book, I will use the term "logical structure" to describe any structure of a subject, whether in a textbook or a classroom, that requires you to learn all of the territory before you can get anywhere in the territory (e.g., solve problems).

Advantages of Logical Structure

The main advantage of logical structure is that when and if you have mastered it, you don't have to worry about classes of problems! To answer a question, any question, in the subject, you simply need to reason from what you have learned — from the "map" you have built in your mind.

Disadvantages of Logical Structure

The disadvantage of logical structure is that for the student — or, in general, the person trying to apply the subject to solve problems — it's an all-or-nothing proposition. Since he or she doesn't know what kinds of questions will be asked, or can be asked, about the diagram (what classes of problems are to be solved), he or she must somehow make a mental copy of the entire diagram. He or she needs to know everything in order to do anything.

But suppose the diagram is huge — suppose it fills a piece of paper the size of a football field. Suppose the student is sorely pressed for time, and is not extremely skillful at reconstructing diagrams from verbal descriptions. Suppose that, in truth, the student doesn't need to be able to answer every question that could be asked about the diagram. Then perhaps he or she might wish there were a better way of learning to solve problems than the all-or-nothing approach of mastering the entire picture. And there *is* a better way, one which I am calling "problem-solving structure".

Problem-Solving Structure

Whereas logical structure shows how the subject is made, problem-solving structure (there are no doubt many such structures, although this book will only describe one) is designed to enable students and other users of a technical subject to solve problems knowing *as little as possible about the subject*, but at the same time enabling them to find out rapidly whatever they want to know — the meaning of a symbol or term, the theorems associated with a given concept, techniques for proofs. A problem-solving structure is organized around the *tasks*, *the questions*, the *classes of problems*, that students and other users of the subject most commonly need to apply or answer. In other words, in a problem-solving structure, we attempt to put at least some *intelligence in the structure* — not merely knowledge, as we do in the case of logical structure, but *intelligence*: aid in solving problems.

The classes of problems addressed by an Environment are those that are described under "The Principle Classes of Homework and Exam Problems" in chapter 1. An Environment is, among other things, an implementation of the professor described under "Semantics versus Syntax" in chapter 3. Thus, an Environment can be thought of as a "soft" calculator — "soft" in the sense that it provides merely heuristic, as well as algorithmic, aids for solving problems, and "soft" in the sense that paper implementations are constructed of soft material.

Problem-solving structure is designed for "just-in-time learning". The term is derived from the term "just-in-time inventory" which, in manufacturing, means that the parts required to build larger assemblies are not kept on the shelves for weeks or months until they are ready to be used, but instead are delivered just a few days before.

Thus the expense of storage is reduced. The analogy in learning mathematics is that, instead of having to store all those parts of the subject in your brain (theorems, proofs, definitions, explanations) just so that you will have the one or two ready when you need it, you store all this in a special kind of document called an "Environment".

Of course, if we think about how we use a traditional textbook as a *reference*, we realize that we often use it in ways other than its logical structure implies. We do not read in one direction only, i.e., from front to back, but rather we often *refer* to various parts of the book: we try to look things up, we refresh our memories because we have forgotten details or even major points, we search for certain kinds of information, we use the book for brief periods of time, then go on to other things, then come back to it. We feel under no obligation to derive the answer to every question by proceeding through the book from the beginning to the place where the answer is given (nor does an expert typically come up with an answer by such a linear process); we try to look the answer up, treating the book (through the table of contents and the index) as a kind of table — a *database*, a means of rapidly solving problems.

Advantages of Problem-Solving Structure

The advantages of problem-solving structure are:

- It allows you to solve problems without having to learn any more than you need to solve just that problem;
- It is designed for speed of finding information.

Disadvantages of Problem-Solving Structure

The disadvantages of problem-solving structure are:

- The organization is unfamiliar to students and other users;
- At present, math teachers and professors regard it as a threat to their traditional ways of teaching (and their teaching jobs), hence are not willing to teach courses based on problem-solving structure. However, my goal is *not* to put math professors out of business, it is to make their jobs easier. It is to render unto the book (and the computer) that which can be so rendered, and thus allow math professors to have more time for their own research, and for helping students whose difficulties cannot be overcome by a new form of presentation of mathematical subjects.
- At present, you have to build your own problem-solving structures (in this book, called "Environments"), as opposed to being able to buy them ready-made. This will soon change.

Logical Structure vs. Problem-Solving Structure: Two Analogies

A good way to contrast the difference between logical and problem-solving structure is to think of the way a colored image appears on your computer screen (in the early 2000's). Typically, you see a blurry approximation of the entire image; then, as time passes, the image becomes less blurry, until you finally see it in its full clarity. *From the very start, you see the entire image*, albeit indistinctly at first. In effect, problem-solving structure is like this, except that you can zoom in on any particular part of the blurry image at any time and obtain the degree of clarity you need to solve your problem.

Now suppose that instead of seeing a blurry approximation to the entire image, you first saw a perfectly clear part of the image, say in the upper left-hand corner of the frame that is to hold the image. Then, gradually, slowly, the perfectly clear part would spread, until you had the entire image before you. Observe that here it would take a long time for you to have *any idea* of the general features of the image; you would have no idea what to expect in the bottom half of the image once the top half was completed. You would have to wait for the image to be fully, completely, displayed, before you could be sure what exactly the image contained. This way of presenting an image corresponds to logical structure.

We are now ready to learn the details of building an "Environment".

Definition of "Environment"

An Environment is a particular kind of problem-solving structure for a mathematical subject — indeed, for any technical subject. An Environment is a set of notes that you develop while sitting in class and/or while doing homework problems. Its purpose is to increase the speed *at which you become able to solve problems* and to save you as much time as possible when you come back to parts of the course in future courses. The goal is that, at all times, you will only need to read what you need to read to solve the problem you are working on. In addition, as you will see, an Environment enables you to understand a subject much faster, and takes you a step closer to knowing a subject backwards and forwards, inside and out.

Of course, the presentation of a subject *x* in a traditional textbook can also be regarded as a problem-solving structure. A classroom course, with or without a textbook, can be regarded as a problem-solving structure. (If you need to solve a problem in subject *x* which you have never studied, you can take a course (if you can find one and if you can obtain permission to take it), and, starting at chapter 1 of the textbook, keep studying until you find out how to solve that problem.) A student's notes can be regarded as a problem-solving structure. A popularization is a problem-solving structure, but one that is very limited in the classes of problems it enables you to solve. What distinguishes an Environment is that it is an *efficient* problem-solving structure, in a sense that is made clear under "How the Environment Idea Should Be Tested" on page 130. In this book, the term *Environment* will always mean *efficient Environment*. There are several ways to implement an Environment. One is by using pencil and paper only. Another is by using a word-processor. A third is by actually creating a computer database. In this book, we will be concerned only with the first two ways. Furthermore, we will be concerned only with *partial* Environments, because implementing a complete Environment for a subject is more work than you will have time for. Complete Environments will someday be available in book form (the author is working on one for vector calculus) and then in database form. Let me say a few words about a database approach to mathematics, because this concept is important.

Databases are computer software systems that are now used throughout the developed world. They are found in just about every large business and, of course, throughout government. A typical business database includes, among other things, information about employees, and permits a manager in the Human Resources Dept. to make requests like, "Give me a list of all employees who are over 30 and who have been with the company at least five years." Or, "Give me a list of all employees who are not managers but who earn more than $60,000 a year." There is absolutely no reason why this powerful technology cannot be applied to the body of "data" which is

a mathematical subject. You, the "user" of the subject, whether you are a student or a person working in industry or a researcher, will one day be able to access a database for any math subject. You will be able to ask questions like, "Give me a list of all theorems and lemmas whose statements contain the terms x, y, or z, or their synonyms." Or, "Give me the definition of the term (symbol) u." Or, "Give me a brief description of the major proof techniques used in this subject." In other words, you will be able to ask most of the basic questions described in "The Principal Classes of Homework and Exam Problems" on page 19.

Fortunately, and remarkably!, there is a way of approximating such a data base using only a word-processor. It is called a *KWIC index*. "KWIC" stands for "key word in context". In essence, it is an index in which each entry is repeated under all the key words in the entry. Here is an example from Linear Algebra. Symbols are not underlined, to avoid any confusing of them with vectors. The page reference for further information about each entry follows the entry, in the form (M, n), where M denotes Saunders Mac Lane's book, *Mathematics: Form and Function*[1], and n is a page number.. Each symbol or commonly-occurring sequence of symbols, e.g., A^TA, that is peculiar to the subject being indexed is in the index.

— A —

A — common symbol for matrix. (M, 199)

A square matrix A is orthogonal iff
$A^TA = I = AA^T$,
i.e., its transpose is its inverse. (M, 199)

— E —

E — common symbol for inner product vector space. (M, 197)

— I —

I — a commonly used symbol for the identity matrix (1s down the diagonal, all other elements 0) (M, 199)

An orthogonal transformation is a function T: $E \rightarrow E$ which is linear and which preserves the inner product, in the sense that $Tu \,.\, Tv = u \,.\, v$ for all pairs u, v. (M 198)

1. Springer-Verlag, N.Y., 1986

A square matrix A is orthogonal iff $A^TA = I = AA^T$, i.e., its transpose is its inverse. (M, 199)

— M —

A square matrix A is orthogonal iff the corresponding transformation is orthogonal, relative to the standard inner product. (M, 1990

A matrix A is orthogonal iff its columns are normal and orthogonal. (M, 199)

A square matrix A is orthogonal iff $A^TA = I = AA^T$, i.e., its transpose is its inverse. (M, 199)

— N —

A matrix A is orthogonal iff its columns are normal and orthogonal. (M, 199)

— O —

An orthogonal transformation is a function T: $E \rightarrow E$ which is linear and which preserves the inner product, in the sense that $Tu \,.\, Tv = u.\, v$ for all pairs u, v. (M, 198)

If T carries any one normal orthogonal basis of E into a normal orthogonal basis, **then** T is an orthogonal transformation. (M, 199)

A square matrix A is orthogonal iff the corresponding transformation is orthogonal, relative to the standard inner product. (M, 199)

A matrix A is orthogonal iff its columns are normal and orthogonal. (M, 199)

A square matrix A is orthogonal iff $A^TA = I = AA^T$, i.e., its transpose is its inverse. (M, 199)

— T —

T — common symbol for linear transformation. (M, 188)

A square matrix A is orthogonal iff $A^TA = I = AA^T$, i.e., its transpose is its inverse. (M, 199)

In my opinion, a KWIC index is a *necessary accompaniment* to a formidably-difficult textbook like Allen Hatcher's *Algebraic Topology*[1], with its many hundreds of

terms and symbols, especially function symbols, that are peculiar to the subject. The existing six-page index is a disgrace, and there is no index of symbols. The book is quite obviously constructed according to the prevailing rule in the Culture, namely, that the purpose of higher mathematics is to separate the Winners (those who can, and are willing to, master mind-numbing presentations like Hatcher's book) from the Losers (those who can't). The book is a prime example of the racket that is university mathematics teaching in the present day.

The longer I work with Environments, the more I believe that a database for each mathematical subject (or a KWIC index approximation to one) is where we are headed in a world in which more than 200,000 new theorems are published in all subjects each year. I believe that one day it will be recognized that *a database is the level playing field for all subjects involving mathematical proofs*. A database *eliminates* the kind of intellectual work that at the time of this writing is considered to be a major part of learning a mathematical subject, namely, memorizing definitions of terms and symbols, somehow keeping straight in one's mind what is related to what, memorizing at least the statements of theorems and lemmas, and accomplishing all this at the pace set by the professor.

However, let me emphasize that a database is part of, but not all of, an Environment. The Environment template for each entity is necessary for rapidly gaining a conceptual view of each entity. (The template is described below under "The "Universal Template for Mathematical Entities"" on page 75.) *The database is mainly an aid to doing and understanding proofs*.

Also let me emphasize that these data bases are *not* Artificial Intelligence (although some math databases may contain reasoning modules that enable you to say things like, "Spend five minutes or so trying to prove *w*, starting from *v*.") The database simply returns information that already exists, but does so very rapidly. It does not "figure things out" except insofar as carrying out logical operations (and, or, not) is figuring things out. Most expert database programmers would probably consider that the only problem with putting, say, vector calculus or statistics or automata theory in a database is that it is not sufficiently challenging. They would admit it involves a lot of work, but given the Environment structure described in this book, there is very little creative intellectual work for them to do.

So that is where we are going. It is inevitable, if for no other reason than the huge explosion in mathematical knowledge. It is absurd to imagine that persons in the future

1. Cambridge University Press, 2002

who need to learn or re-learn how to solve a certain type of mathematical problem, will find it most efficient to sign up for a course (assuming one is being offered), still less wait until the next semester when it will be offered.

However, until math databases (Environments) become widely available, you, the student will have to create your own Environments, and these will necessarily be partial ones.

At this point, you might want to take a few minutes to look at some of the examples of partial Environments in the appendices.

:The Environment concept, of course, is not entirely new. The following are a few precursors.

- *The Universal Encyclopedia of Mathematics* (Simon and Schuster, N.Y., 1964), which covers a variety of high-school and early college-level subjects in geometry, trigonometry, elementary calculus, etc.;
- the *Mathematics Dictionary* ((ed. James and James), D. Van Nostrand Co., Inc., N.Y., 1959), which covers numerous terms typically confronted in high-school and undergraduate college math.
- the *Encyclopedic Dictionary of Mathematics* (ed. Iyanaga and Kawada, The MIT Press, Cambridge, Mass., 1977; later edition published in the early nineties), which covers much of higher mathematics. If you are planning to get a Ph.D. in mathematics or in the more theoretical branches of computer science or in physics, I think the *Encyclopedia Dictionary* (EDM) is indispensible, despite its cost (more than $300 new in the early nineties, though used copies can be bought for as little as $50). The effectiveness of several ideas to be described below can be seen in actual practice in this book: e.g., the marking of every term whose definition and context can be looked up, via the index, every time the term occurs (in other words, a paper implementation of hypertext); the separation of proofs from the presentation of theorems (no proofs are given in the EDM); the fact that every symbol can be looked up. In the 1977 edition, you can find the major theorems and ideas of point-set topology in 11 pages; of linear algebra in 10 pages; of probability theory in six pages.

However, the Environment idea set forth in this book involves many more features than are contained in the above books.

The Structure of an Efficient Environment

The question now is, "How do you structure a subject for efficiency at solving certain classes of problems?" The answer is given in this section. It applies not only to any mathematical subject, but also to any technical subject, including the use of computer programs (see Schorer's *How to Create Zero-Search-Time Computer Documentation*). You can implement the ideas in this section — in fact, in this book — with pencil-and-paper, word-processor, or database. Your goal is to put at least some intelligence *into the Environment*, not merely the knowledge that makes up the subject, but intelligence — aid in solving problems.[1]

It may help you to better comprehend the nature and value of the Environment idea described below if I exaggerate a little: the form of an efficient environment (including the Universal Template for Mathematical Entities that is described below) is a *variable* whose domain of values is *mathematical subjects*. We "plug" a mathematical subject "into" an Environment. We want to make (to whatever extent possible) all mathematical subjects look "the same", because in a time when more than 200,000 new theorems are published *each year*[2], we feel that it is essential that users of mathematics (including mathematicians) be able to rapidly find the material they need in a subject with which they are not familiar.

Title Page

It probably goes without saying, but let me say it anyway, for the sake of completeness: the first page of any Environment is the title page so that, a year or two or ten years from now, you don't have to spend time figuring out what exactly the Environment covers. This page contains just the title of the subject and the date you began building the Environment.

Start Page

Every Environment has a Start Page giving you a global view ("Big Picture") of the subject, which means, a summary of *the major classes of problems that can be solved in the subject*. The Start Page is the second page of the Environment. In a

1. The slogan, "Put the intelligence in the Environment!", was originally that of a marketing manager in a computer company, in reference to the user Environment for a new software product. I do not know the manager's name. But the purpose of his slogan was to inspire programmers (and technical writers) to reduce the amount of time that users had to think about, and study documentation about, how to accomplish what they wanted to accomplish using the software.

2. Ulam, S. M., *Adventures of a Mathematican*, Charles Scribner's Sons, N.Y., 1976, pp. 287-289

complete Environment, namely, one you would buy ready-made — the author of this book is in the process of developing such an Environment for the subject of vector calculus — it must be possible to get to *every* problem-solving procedure in the Environment by following references from the Start Page. Normally, in an Environment you create for yourself, you will not have the time to achieve this degree of rigor.

The Start Page answers the question, in our list under "The Principal Classes of Homework and Exam Problems" on page 19, "What is the Big Picture of this subject? What does the forest look like, apart from the trees?" The format of the Start Page is the same as that for the "Universal Template for Mathematical Entities" described below, except that some categories are not present.

The Start Page lists the major operations you perform on the principal entities in the subject. Examples of entities are: numbers (of any kind), functions (of any kind), spaces or sets (of any kind), e.g., groups, rings, fields, topological spaces, vector spaces; also physical entities, e.g., force fields. Just about any noun pertaining to a mathematical concept can be an entity.

In elementary calculus, the principal entities are the real numbers and continuous functions on the reals. The operations you perform are: finding the derivative of (some) continuous functions, finding integrals of continuous functions; solving certain classes of problems, e.g., finding areas, volumes, centers of gravity. As you start constructing and using Environments, you will find yourself wondering about applying it even to — *especially* to — subjects which have a reputation for being notoriously difficult. For example, you might ask a physicist what the principal entities are in quantum mechanics, and what the principal operations are that are performed on them.

You need not complete your Start Page before continuing with the rest of your Environment, but you definitely need to keep asking yourself, as you proceed through the math course, "What are the basic entities in this course, and what operations do we perform on them?"

After the Start Page comes the body of the Environment. All topics are in alphabetical order so you can find things *fast*.

The "Universal Template for Mathematical Entities"

The Universal Template for each entity typically occupies a page or at most two in hand-written implementations. It makes all entities "look the same" from a problem-solving point-of-view, which means that learning about and understanding each new entity is easier since, to a certain extent, that entity is similar to other entities you have learned about. The Template consists of the following categories. Each category is

designed to answer one or more of the questions under "The Principal Classes of Homework and Exam Problems" on page 19. (The questions regarding proofs are answered in the chapter, "Proofs".) Each category is explained in detail below.

- **Definition and purpose of the entity**

- **Ways of representing the entity**
  Mappings between representations

- **Common operations on the entity**
  Determining equivalence of entities
  Arithmetical operations on the entities (not applicable in all cases, of course)
  Identifying the type of an entity
  Determining basic building blocks of the entity
  Breaking an instance of an entity into sub-entities
  Making more of the entity

- **Properties of the entity**

- **Functions from the entity to the same or other entities**

- **Theorems on the entity**

- **Types of the entity**

- **Related**

If you are thinking, "I haven't time to enter all that information!", let me first remind you that, in the course of taking class notes or your own study notes, you *will* write down at least some of this information. So it's just a question of *where* you choose to write it down. Second, you don't have to make a complete Template for *every* concept you come across; only for those which are difficult for you. Third, you can always simply reference parts of the textbook when you fill in the details for each referenced section. You can reference pages that at least give examples of solving a certain kind of problem even if they don't give you the complete method. You can reference pages that give proofs of theorems. Then, as you figure out a complete method, or work through a proof, you can reference the pages in your course notes where you wrote up the method or the proof. Fourth — and this is important — the

mere discipline of dividing up the information on each concept in accordance with the categories of the Template, *will itself* help you understand the concept.

Fifth, **you can make a first approximation of an Environment by simply creating an index using the above template.** (You will need a computer word-processor to do this.) Following each index item, e.g., "Properties of ...", you will give the page no. or nos. of that material in your textbook (and/or in other textbooks). You can then expand on specific topics in a separate, alphabetically-arranged, volume of notes.

We now consider each heading in detail.

Definition and Purpose of the Entity

Here you put the formal definition of the entity (or a reference to the formal definition in the textbook), along with additional notes, comments, to aid your intuition — what a sympathetic grad student might tell you about the entity during a walk across campus. For example, following the formal definition of the entity, "continuous function", you might have a note and diagram showing that a continuous function has "no gaps". For the entity, "topological space", you might have words to the effect, "A topological space defines the *relative nearness* to each other of its elements." In many cases, of course, your definition will contain other terms that are part of the subject, and the definition of those will contain other terms, etc. This recursive structure in fact keeps the length of each definition manageable, and if you can always look up the meaning of a term rapidly (as you can if all terms are in alphabetical order) then it does not require you to memorize the meanings of the terms as you come across them.

Include a few notes on the history of the entity if you can! The history will often do wonders for making a concept seem less forbidding, less remote. (Personally, I have no hesitation in saying that, If I don't understand the motivation for a concept — what led to its discovery — then I don't understand the concept.) When did the entity first become important in mathematics? What questions, problems, was it a response to? Why does it have the name it does? (If the name is not obvious, this information is usually difficult to come by, yet having it can do wonders for helping your understanding.)

This category in the Template answers the question from our list "The Principal Classes of Homework and Exam Problems" on page 19, "What does the term (or symbol) x mean? If x represents a difficult concept, what are some guides to improving my intuitive understanding of x?"

Ways of Representing the Entity

The representation issue is not equally important for all entities, but it is important that you know when you are, in fact, confronting this issue. Probably the most familiar example is the various ways of representing a complex number, i.e., by rectangular coordinates, polar coordinates, a vector, or by stereographic projection. In computer science, the issue of representation is particularly important, since different types of representation — e.g., of various data types, e.g., arrays, decimal numbers, linked lists — may result in different computation speeds.

Closely related to representation, is the issue of "decoding". It is often the case that a concise mathematical description (an "encoding") is not at all a description that is easy to understand the first time you confront it. So an important question to ask when you confront something difficult is whether the difficulty is inherent in the concept, or whether it lies in the decoding effort required.

.**Mappings between Representations**

If an entity has several representations, then an immediate question is, "How do we convert from one representation to another?" For example, in the calculus, we often need to convert a representation of a curve in one set of coordinates, e.g., Cartesian, to another, e.g., polar.

Common Operations on the Entity

In mathematics, the term "operation" is often restricted to mean "function whose domain is itself a set of functions". However, in this book, the term means any kind of function, formally or informally defined. It means a *task* that you the user of the Environment can perform. I say it without a moment's hesitation: *Task-orientation is the great vacuum cleaner of mathematical (indeed of all technical) presentations*. In times of seemingly hopeless difficulty, if you ask yourself, "What tasks are performed in this subject (or part of the subject)?"), you will find that doors to understanding begin to open.

The tasks (operations) on the typical entity in an Environment include:

- operations on the real numbers include addition, subtraction, multiplication, division, extraction of roots;
- operations on continuous functions include finding the derivative and the integral;
- operations on polynomials include factoring, converting to a standard form, adding, subtracting, multiplying, dividing;

- operations on equations include adding a term to both sides; multiplying through by a term; determining if the equation is solvable and, if so, how many roots it has; solving the equation;
- operations on expressions in boolean algebra include simplifying, computing the truth value;
- operations on vectors include finding the length ("size") of a vector; adding, subtracting two vectors; multiplying two vectors (scalar product and cross product);
- operations on infinite series include determining if a series converges and, if so, to what number; adding, subtracting, multiplying series; determining the coefficients;
- operations on computer programs include running the program; proving its correctness; debugging it; proving that the program is in a certain class of programs; determining the computational complexity of the program; determining if its computations are time-bounded or space-bounded.

This category of the Template helps you to answer the following questions from the list under "The Principal Classes of Homework and Exam Problems" on page 19: "What are the basic operations we perform on the entity *x*?"; "How should I go about solving a problem of type *w*? What are some good ways of approaching this type of problem?", and the sub-class of this class "What is the formula for *y*?"

The idea of organizing the information about each entity by the operations that can be performed on the entity lies at the heart of the Environment concept. In computer science, the idea has been well-known since the seventies at least, under the name of *abstract data structure*. Here is the background.

A data structure, as its name implies, is a way of organizing data. For example, a list, (*a*, *b*, *c*, *d*, ..., *z*), is a data structure, the items of data being represented by *a*, *b*, *c*, etc., and the address of any item of data being specified by a number, *k*, representing how far down the list, from left to right, you need to count in order to reach that item. An array is a data structure, with each item of data having a row and column address.

Such structures are indispensable to programmers, but, as programmers gradually found out, there were many ways to implement each data structure, depending, for example, on the programming language they were using and the constraints of program operating speed they were faced with. For example, a list could be implemented as a one-dimensional matrix if the language contained the matrix data type, or it could be implemented as a linked list, meaning a series of memory locations each (except for the last) containing the address, in computer memory, of the next location. It slowly dawned on programmers that it didn't really matter how the data was *stored* in the computer; what differentiated one data type from another was the *kind of operation* you had to perform to access the data in the structure. Thus, the

programmers argued, you have a list data structure when you can access any piece of data in it by a command (operation) which in effect says, "Get me the *k*th item in the structure." You have an array data structure or table when you can access any piece of data in it by an operation which says, in effect, "Get me the data in the *i*th row and the *j*th column of the structure." It doesn't matter, as far as the abstract definition of the data type is concerned, how this is done.

So the key idea here is that the data type *is defined by the operations that can be performed on it*. Period.

We adapt this idea to Environments by requiring that every mathematical entity in the subject — I am speaking of a complete Environment here: you will not have time to do this in the Environments you build for yourself — have associated with it, in the Environment, *all and only* the elementary operations that can be performed on the entity, with an explicit reference to the location in the Environment where the procedure for performing the operation can be found.

Determining Equivalence of Entities

One of the most important common operations is determining if two entities are equal, or, more generally, "equivalent". Thus, if the entities are complex numbers, they are equivalent if their real parts are equal and their imaginary parts are equal. If the entities are sets in the subject of modern algebra, then the sets are equivalent if there exists a type of function called an "isomorphism" between them. On the other hand, if the entities are topological spaces, then the spaces are equivalent if there exists a type of function called a "homeomorphism" between them. If the entities are finite-state machines (which are studied in the computer science subject, automata theory), then the machines are equivalent if they generate the same language, i.e., set of strings over an alphabet.

One of the most helpful first questions you can ask about a new and seemingly-difficult-to-understand mathematically entity is, "How do I tell if two of these entities are 'equal' (i.e., equivalent)?" Of course, in some cases, the entity may be the equivalence relationship itself, as in the example of congruence in the Appendices.

Building Blocks of the Entity

"Building blocks" are "simple" elements out of which we can make more complex elements. For example, in linear algebra, we learn that a set of linearly independent vectors can be used as the buildings blocks of a vector space. In topology, we learn that the elements of a sub-base can be used to define the topology of a space. In symbolic logic, we learn that the two logical functions *not* and *and* can be used to

construct any logical function we wish. In group theory, we learn that, in certain groups (namely, free groups), there exist elements which can be used to generate the entire group.

The building blocks concept is an example of "Fundamental Concept 2: Structure, or Breaking Complex Things into Simpler Things" on page 47.

In passing, I should mention that not all entities have building blocks! In this case, they may be described as having the property of being indecomposable. A simple example is the prime numbers, like 2, 3, 5, 7, 11, which can be thought of as having no building blocks (factors) except themselves and the number 1, whereas composite numbers, like 4, 6, 8, 10, have other building blocks (factors).

Breaking an Instance of an Entity into Sub-Entities

Thus we break a a set into various kinds of subsets — for example, a group into various kinds of sub-groups, a function into various kinds of sub-functions, a space into various kinds of sub-spaces, etc.

Making More of the Entity

A related common operation for many entities is that of making more of them out of one or more given ones, e.g., beginning with a group, making another group out of the set of automorphisms on the given group; or given a finite set of topological spaces, making another topological space out of the product of the spaces.

Properties of the Entity

This is a redundant category, since ideally it is covered by the categories "Theorems on the Entity" and "Types of the Entity". (Properties are typically established by theorems and lemmas; an entity with a certain property is a type of the entity.) But often it will help your understanding if you have this separate category. You can jot down the properties here as you come across them in the text or in lectures. In fact, it is a good habit to get into to ask, of each new entity, "What are the properties of this thing?"

Functions from the entity to the same or other entities

In algebraic subjects, the most common of these functions are homomorphisms and isomorphisms. Probably the best idea is to have a page headed, "Functions, types

of", and then on that page, for each entity, an item, "[name of entity], functions on", with an indication that details on each item can be found under the item.

Theorems on the Entity

Theorems assert facts, properties, about entities. This category is where you *list* all the theorems (or references to them in your textbook) that you come across that pertain to the entity. *No proofs!* After each theorem, you add a reference to where the proof (or *a* proof) can be found, e.g., in a "Proofs" section at the end of the Environment, or in the textbook. The chapter, "Proofs", has a number of guidelines for writing out proofs which you will find helpful. If you can understand the textbook presentation of a proof without great difficulty, then the reference is simply to the pages in the text. If you find it necessary to redo the proof in structured form, as described in the chapter, "Proofs", then your structured version goes in the back of the Environment.

When there are many theorems associated with a given concept, it will be important to group them in a way that will help you to *quickly* find the theorems you need in the course of doing proofs. Thus, e.g., in the elementary number theory Environment of which a part is given in "A Number Theory Environment (partial)" on page 203, theorems concerned with moduli are divided into:

- those concerned with prime moduli, p;
- those concerned with moduli which are powers p^k of a prime p;
- those which are concerned with moduli which are products of different primes to arbitary powers, ${p_1}^k {p_2}^m ... {p_n}^t$.

Note: Whenever you see an exercise at the end of a textbook chapter that asks you for a proof of something, you should add that something — or a reference to it — to your list of theorems under the appropriate Template. That's right: if an exercise states what amounts to a theorem, then it belongs in your list of theorems for one or more concepts. If you doubt the importance of this, consider how often have you come across a statement like this in a textbook: "Now to prove this theorem, we make use of the result achieved in problem 47 of chapter 12..."? Or, even worse, how often have you slaved over a problem, then found, in the Answer section, that it is very easy if you happen to have worked a problem a few chapters earlier, and very difficult if you haven't? By now, I hope that you are beginning to develop a healthy skepticism about the pedagogical benefits of such textbook practices.

Another benefit of listing and, possibly, grouping theorems is that it discourages a bad habit which some math students get into. These students are typically those for whom math is first and foremost a mechanism for judging their own worth — not

merely their own worth as math students, but as human beings. They are tormented by thoughts like, "If I were any good, I'd be able to solve this problem in a few seconds!" "If I were any good, I'd be able to do this proof without looking at the book!" In the latter case, what they often wind up trying to do is to do the proof from "first principles", i.e., from the axioms and definitions which they remember.

Now the truth is that it is seldom the case that a difficult proof deep into a subject can be done from first principles. That is one reason why the theorem appears later, rather than earlier, in the textbook. You will, in fact, be saddling yourself with proving all the lemmas and theorems (the equivalent of subroutines or procedures in a computer program, per our discussion under "Representation" on page 46) which you need to construct your proof. (Most of these lemmas and theorems have already been presented in the text). Or, what is more likely, you will be condemning yourself to failure and further self-recrimination.

Your aim in developing a proof (and, indeed, in solving any problem in math!) is to get the most for the least work. Remember that you can use any theorem or lemma that is logically prior to the one you are trying to prove, *whether or not you have worked through the proof of that theorem or lemma*. Lists of lemmas and theorems with mere references to the proofs, are a way of encouraging you to develop this very important habit, and to get out of the habit of tormenting yourself with, "If I were any good..."

This category answers the question, in our list under "The Principal Classes of Homework and Exam Problems" on page 19, "What does the theorem or lemma called *y* assert?"

Incidentally, in the case of theorems with names, e.g., Rolle's Theorem in the calculus, you enter the name alphabetically and then reference where it is listed and/or proved in your Environment (or in the textbook).

Types of the Entity

This is where you make a list of all the different types of the entity as you come across them in your reading. You will be amazed at how much this helps your understanding of the entity. Of course, the proofs that the hierarchical relationships in fact hold is another story, and will have to be done by the textbook author and/or by you. Kit Carson's motto was "First get the lay of the land." and that is exactly what finding out the various types of an entity, does. Whenever possible, you should try to indicate which types are subsets of other types, using any convenient means, e.g., indenting each sub-type under its type, or using Venn diagrams. In my opinion, it is outrageous that no representation of the types of each entity are not a normal part of

every textbook Careers are built on the labor of transmitting this knowledge via words alone. A simple picture, perhaps a hierarchy of diagrams, would save you, the student, many hours.

Here is an example of a Types of Group list:
Abelian (commutative) groups,
free groups,
cyclic groups,
finite groups,
infinite groups,
continuous groups,
quotient groups,
symmetric groups of degree *n*,
alternating groups,
solvable groups,
simple groups,
groups of automorphisms,
and the various types of sub-groups, e.g., normal subgroups, centers, etc.

Examples of the entity rightfully belong here, although you may want to break this rule intelligently now and then. Thus, e.g., in most group theory textbooks, following the definition of a group, the example of the integers is given, with addition as the group operation. But this group should also go into your Types of Group list.

One of the most important Types of pages in any Environment is "Types of function". Keeping track of the important types of function in a subject (with sub-types clearly indicated when you happen to know which are, in fact, sub-types) is in itself an aid to understanding, as you will find.

Related

The term "Related" (which you will probably soon start abbreviating to "Rel") means other entities which are closely related to the one under which it appears. Its purpose is to help you in the course of problem-solving searches. This category can be named "See Also" if you prefer.

This category answers the question, in our list under "The Principal Classes of Homework and Exam Problems" on page 19, "What theorems or lemmas are closely related to theorem or lemma *z*?"

And that's it! You keep adding your notes in this form, and improving them as you proceed through the course.

Let me hasten to say again: you don't have to write out things if you are pressed for time. You can, at least as a start, reference pages in the textbook. But do it, so you don't keep have to do those linear searches through the book if it has a bad index, and it probably does. Your goal is to reduce the amount of intellectual labor you have to go through in solving problems — in short, to make problem-solving *as much a matter of looking up things as possible*. Your goal is to reduce the amount of memorization you have to do *in order to access information*.

"Einstein claimed never to memorize anything which could be looked up in less than two minutes." — "Albert Einstein Anecdotes", Google, 9/22/18.

This does not mean that you shouldn't memorize material! If you are forced to take closed-book exams, you will have to memorize some of the content of the course. But you should never have to memorize because the information is so hard to find in the text or your notes that the cost in time and effort would be too great not to memorize it. Paradoxically, as you will find, one of the appealing things about an Environment is that it makes memorization much easier.

Why Go Through the Trouble of Building an Environment?

Let me anticipate a few of your possible initial objections to building an Environment for a mathematical subject:

• *"I have no time to produce such careful notes!"*

But when you are in class, you *will* take notes *of some kind*. When you work through passages in the text, you *will* take notes *of some kind*. Why not do the job once, in a way that will save you almost all duplication of effort, and will at the same time improve your problem-solving ability?

Furthermore, and I must emphasize this point, you don't have to apply all the ideas in this book! In fact, you will find, after you have built Environments for a few subjects, that the Environment form becomes one that you keep in your head, a way of thinking about each subject, a kind of Cheshire Cat's grin without the Cat:

"'All right,' said the Cat; and this time it vanished quite slowly, beginning with the end of the tail, and ending with the grin, which remained some time after the rest of it had gone."

— Lewis Carroll, *Alice's Adventures in Wonderland*, in *The Annotated Alice*, ed. Martin Gardner, New American

Library, N.Y., 1974, pp. 90-91.

• *"I can't believe that simply reorganizing the information which constitutes a mathematical subject will improve my math grades."*

The only reply I can make here is — and it is based on years of experience — *try it!* At the least, start observing, and keeping track of, what you spend your time on in the course of working problems, particularly hard problems.

• *A complete Environment is too big! It has too many pages!*

That is because an Environment reveals, probably for the first time, the true complexity of mathematical subjects. A great deal of knowledge surrounds even the simplest concepts (entities), namely, the knowledge represented by all the categories in the Universal Template. Professors, both as teachers in the classroom and as authors of textbooks, attempt to conceal this complexity ("the student should know", "the student should be able to figure out..."). An Environment makes it explicit and, in the process, gives you a systematic way of dealing with it.

• *"Putting mathematical knowledge in alphabetical order seems silly."*

Or, as one student expressed it, in a masterful application of the rhetorical device known as *tmesis*, "I haven't time to put things in no alpha-bloody-goddamn-betical order!" A mathematician once remarked to me, with ill-concealed contempt, "What do you think mathematics is, some kind of alphabet soup?"

My reply to both comments is that alphabetical order is simply a means of obtaining random access of information in textual implementations of Environments. In a database implementation, there would be no need for storing the information alphabetically.

Now let us proceed to a discussion of some of the techniques you can use in building an Environment.

Guidelines for Building Environments

Build the Smallest Environment You Need

Throughout this book, I am presenting the elements of a *complete* Environment. You will probably never have time to build a complete Environment for a subject. But there are many ways to implement partial Environments. For example, you may find

that the content of a subject is not so hard to understand. Then you may be able to get by with simply adding entries to the index of the textbook. Or you may find it necessary to use the Universal Template described above only for a few difficult concepts.

But always remember that, no matter how you choose to implement your partial Environment, you should explicitly state, e.g., on a page referenced from the Start Page, how the partial Environment "works", i.e., the general rule to follow in attempting to solve any problem in the subject. The rule may be no more than, "Try to solve problem using index of textbook; if not successful, then use this partial Environment." You can, of course, add references in the textbook pagesto rewrites of certain difficult theorems which you have included in the partial Environment.

Probably the simplest approach to building a partial Environment is:

1. Simply look at the examples in the Appendices of this book and take all lecture notes and personal study notes according to the format shown there. Refer to "The Structure of an Efficient Environment" on page 74 for details if you need them.

2. Work all homework problems, and take-home exam problems, *using the Environment*. If the Environment doesn't have what you need to solve the problem, add the necessary information to the Environment, or a reference (e.g., to the textbook page) to where that information is!

It's that simple.

If you want further explanations about various aspects of an Environment, use the Table of Contents and the Index of this book to look up what you want to know about. Remember that there is only one criterion of the worth of the ideas of this book, and of your success in applying those ideas, namely, whether or not your math grades improve!

Use Alphabetical Order

Take all lecture notes and study notes in alphabetical order by mathematical term, as shown in the appendices. Include synonyms. In other words, every time a new concept is introduced in class which you sense will be difficult, every time a new term or symbol is introduced, you start a new page in your loose-leaf notebook (or a new look-up-able heading in your word-processor file). Each term gets a separate page (or a separate heading), so that alphabetical order can be maintained. Every time you come across more information about a concept already in your Environment, you simply enter it under that concept.

Put non-alphabetical symbols at the end in any order you wish. Greek letters can go either where you think they belong in our alphabet, e.g., "Ω" (omega) under "O", or

else at the end. The only condition is that you know immediately where to look them up.

Your goal should be to make the Environment such that all you need to know is the term or a synonym for the term and you can find it! You shouldn't have to *figure out* where information on a given topic must be, e.g., by going through a reasoning process such as, "Well, *x* is a case of *y*, and *y* comes under the general heading of *z*, therefore..."

There is, of course, nothing wrong with marking up the index of the standard textbook and using that as part of your Environment (What vs. How). In my experience, however, indexes are so incomplete that sooner or later you are forced to start creating your own separate pages. Also, there is not enough room in a book index to add additional reference material (see, e.g., "Related" on page 84). Your own experience and needs will be the best guide.

Build As You Go

The idea is *not* to somehow build your Environment *and then* start using it to solve problems. No! You start using it immediately, on the first day of your course, typically, though not necessarily, by making a Start Page. This might entail some persistent questioning of the professor. If he or she is reluctant to give you the kind of problem-solving-oriented Big-Picture information you want, don't annoy him or her, since he or she is the one who gives you your grade. Try to get the information some other way, e.g., by asking another professor, or by looking in encyclopedias or math dictionaries or popularizations, or by talking to students who have taken the course.

As you work problems, ask yourself, first, what class(es) of problem the one you are working on belongs to, and then, what procedure (heuristic) enables you to solve the problem. Write down the procedure in whatever form you feel will be clear to you when you return to this part of the subject later. Remember that the procedure is your goal, because a correct procedure enables you to solve any problem in the class.

The key questions to keep in mind as you build an Environment are:

- What questions (*classes* of problems) do I (and other students) of this subject typically find ourselves asking about it or find ourselves being asked about it?
- How much of the content of this subject can I avoid remembering by putting it in the Environment? Or, in other words, how *look-up-able* can I make the task of solving problems in this subject?

Creating an Environment is a build-as-you-go proposition. You are incrementally *engineering the use* of a mathematical subject for the efficient solving of problems. Every time you *and* the Environment are unable to solve a problem, you enter the missing information (or references to it) in the Environment.

If you are motivated to become as good as you possibly can at mathematics, you should keep a rough record of the kinds of problem-solving activities you spend the most time on. This will enable you to know which skills you need most to develop.

Build Your Environment So That It Allows You to Select the Depth of Detail You Want

This is simply the application of structured programming principles to Environments. (See "Fundamental Concept 2: Structure, or Breaking Complex Things into Simpler Things" in chapter 3.) You organize the material for each topic in the way that will give you the most rapid problem-solving ability with the least effort. You can find all the details you need, but you are not forced to wade through details you are not at the moment interested in.

Use Tasteful Redundancy

You want to save yourself turning pages as much as possible. Therefore, *space permitting*, you should always include brief reminders of the meaning of difficult terms, brief statements of theorems (in addition to mentioning the theorem number), where this can be done without cluttering the page or forcing it to spill over onto a new page.

Have a "Formulas Page"

Rather late in the development of the Environment concept, I realized how helpful it is to have a page of the most frequently occurring formulas — usually, equations that define important functions — that can be referred to at any time. A subject in which this is especially helpful is vector calculus. On the page, closely related equations — e.g., those express some kind of duality in the subject, can be written close to each other.

Aim for "Understanding at a Glance"

View each page of an Environment — or each computer screen, in the case of a computer implementation — as a *gestalt* — a visual whole, not a sequence of words. Think of a page as a *picture*, rather than as a medium for holding sequences of words and symbols. Make the visual layout of each page correspond, as much as possible, to the logical relationships it contains. This is particularly important in regard to proofs, as explained in the chapter, "Proofs".

Here are some ways to make "eye logic" do some of your mental work for you:

• Give a separate line to each item in a list or sequence where appropriate, rather than running them end-to-end, in prose form;

• Use indents to show subsets, special cases.

• Give similar locations and sizes to similar items, e.g., if you use the same geometric figure repeatedly in different drawings, whether in a cinematographic presentation (as described under "Cinematographic Presentation" on page 94) or otherwise, try to keep the size and the relative location within the framing border of the illustration consistent, in order to reduce the necessity of mental translation;

• Decide on one and only one term to refer to the same thing, although all synonyms *should* appear in the Environment. The best idea is to put all the information for a given concept or term or entity under the symbol, *not* under the name of the concept, since the symbol is what will appear most often in text or lecture notes.

• Get straight on the pronunciation of symbols, e.g., Greek letters. If you keep track of what you spend your time on in the course of problem solving, you may be surprised to find that you *do* often interrupt a train of reasoning to try to recall the name of a Greek symbol: "Now, let's see, is that *eta* or *nu*?" In my opinion, it is a worthwhile effort to memorize the names of the Greek letters, once and for all.

• In writing down difficult concepts or lines of reasoning, "cinematographic presentation" is a useful technique, i.e., presentation via a sequence of drawings, as described under "Cinematographic Presentation" on page 94.

• Use tree graphs! The absence of these in popularizations, not to mention textbooks, in which there are numerous entities of the same type — groups, algebras, moduli spaces, topological spaces to name just a few — is proof of the fundamental incompetence of most authors of popularizations and textbooks. Tree graphs show *at a glance* how all the entities *are related to each other.* This is not knowledge that the student should somehow accumulate after weeks, months, of grueling study. It is not knowledge that should be presented in page after page of tedious and mind-numbing prose. It is knowledge that should be on page 1 of the popularization or text.

(You will almost certainly need to ask your professor where in the tree graphs many, perhaps most, of the entities belong. My recommendation is not to do this in class, because he might not know, and hence will have to do a little research on his own, but instead to ask him in office hours. Be suitably humble: "I'm terribly sorry to have to bother you with this, but I have found that, when there are many entities of a given type, it helps me enormously in my understanding, to know how they are related, and a tree graph is an excellent way to show this.")

• Use tables! In fact, a good question to ask yourself when you are confronted with a complicated collection of facts about a given set of entities, e.g., topological spaces and their properties, or abelian groups and subgroups and their properties, is: "Could all this be put into table?" Often you will find that the answer is yes, and that

the labor of constructing the table will be amply rewarded by the increase in your understanding. A table is a prime example of "understanding at a glance".

For more on math subjects as tables, see "Appendix A — Linear Algebra and Tensor Calculus Are Just "Great Big Tables"!" on page 100.

.• Closely related to tables is the practice of putting ***closely related entities on the same page*** even though you may have separate entries for them elsewhere in your Environment. Good examples are the functions *div*, *grad*, and *curl* in the vector calculus. The definitions of these functions are much easier to memorize when they are seen side-by-side. The same is true of closely related theorems, e.g., Stokes' Theorem, Green's Theorem, and Gauss's Theorem. Of course, you must have a reference to this comparison page under the entry for each function or theorem.(

It is a scandal that every calculus text does not present a side-by-side, graphical representations, with words, of all the different types of derivatives.)

• Use drawings and ordinary language, when possible, to tell what the symbols mean. This is especially helpful, e.g., in the three theorems in the preceding subsection, and in Maxwell's equations. Tell, via a reference to a drawing, what each side of an equation means.

Yes, Include Frequently-Occurring Strings of Symbols in Your Environment!

In most subjects, some strings of symbols occur repeatedly. For example, in Basic Calculus and Vector Calculus, among many others, the strings,

$$adx + bdy + cdz$$

$$\iint f(x, y)dxdy$$

In complex number theory, among many others, the strings

$$\sin(\omega t + \varphi)$$

$$e^{2\pi im/n}$$

Why shouldn't you be able to quickly look up frequently-occurring strings of symbols in your Environment for the subject, and there find your notes (with a picture,

if at all appropriate!) explaining what the string stands for, and why it is important, and what it arises from, and/or a reference to the places in the text that explain it.

In some cases, it will be very useful to have, on one page, a list of related strings, e.g., in the calculus, those pertaining to the total differential.

Of course, the term "strings of symbols" includes strings of length one. These are the symbols that textbook authors occasionally (but, shamefully, not always!) put in a glossary of symbols at the back of the book.

In passing, let me confess that it was late in my development of Environments that I realized the importance of being able to rapidly look up strings of symbols. I had until then accepted without thought that, if I had forgotten what a string of symbols stood for, or what it was derived from, the fault was mine: if I had been a *good* student, I would have remembered. I now consider that attitude as ridiculous. If you have forgotten something, then you should be able to look it up rapidly. Period.

Use Drawings!

(A further elaboration on this section is given in the section, "On Drawings" on page 146.)

Most Mathematicians Think Visually

The best mathematician I ever personally knew — a man who had the entire undergraduate mathematics curriculum at his fingertips, along with a vast amount of knowledge pertaining to his specialty, algebraic geometry — this mathematician never used pictures, at least not in any of the conversations I had with him about mathematical subjects. But this is unusual. To repeat an earlier quotation:

"Some mathematicians, perhaps 10 percent, think in formulae. Their intuition deals in formulae. But the rest think in pictures; their intuition is geometrical. Pictures carry so much more information than words. For many years schoolchildren were discouraged from drawing pictures because 'they aren't rigorous'. This was a bad mistake. Pictures are not rigorous, it is true, but they are an essential aid to thought and no one should reject anything that can help him to think better." — Stewart, Ian, *Concepts of Modern Mathematics*, Penguin Books, N.Y., 1981, p. 5.

(See also the remarks by various mathematicians in the chapter, "Proofs".)

Against Print

I am now going to say something shocking: don't repeat it within earshot of any mathematics professor: here it is: I think that one of the worst things that ever happened to mathematics was the invention of the printing press — or, I should say,

the invention of movable type —, and three of the best things were the invention of chalk-and-blackboard, paper-and-pencil and photo-lithography (including the copying machine). The daily work of mathematics is done with pencil-and-paper and/or chalk-and-blackboard. If you ask a mathematician to explain a concept, he does *not* start setting type or cranking up his word-processor. He reaches for pencil and paper or chalk if the room happens to have a blackboard. Whenever he feels the need for a picture, he draws one and labels it as necessary. If he has used pencil-and-paper, then chances are you could take what he has written and copy it (including his drawings), adding clarifications as necessary, go to a copying machine and make as many copies as you need to distribute to other interested persons, and, if your handwriting is reasonably clear, their difficulty in understanding the explanation *will not be because it is handwritten, or because the drawings are free-hand.* (I once read the author's handwritten draft of what became a very important paper in computer science and was amazed at how "intimate" it seemed, how much it seemed to invite me to dive in and understand what was going on. I hadn't the slightest impulse to set it aside and read the published version instead.)

If a quarter of the effort that Professor Donald Knuth put into perfecting the typesetting lanuage TeX had gone into perfecting a program that made the creation and modification of labelled mathematical drawings very easy, the mathematical world would be a better place.

"Technical illustrations...cause no end of trouble. Don [Prof. Donald Knuth] says that the amount of work involved in preparing a paper for publication is proportional to the cube of the number of illustrations." — Knuth, Donald, et al., *Mathematical Writing*, Mathematical Association of America, 1989, p. 40.

We must ask why illustrations cause such a great deal of trouble. Any student or professor of mathematics with a reasonably clear handwriting can write a page of mathematics and draw an illustration free-hand that is perfectly clear and understandable. He or she can then take the page to any copier, and make any number of copies of it. Or he or she can bring it to a local print shop that can do photo-lithography (offset printing — and most of them can) and have the shop run any number of copies. *So what's the problem?* The problem is that the mathematical community believes that anything that is not typeset, and in which the illustrations are not drawn by a professional illustrator, and provided with typeset labels, may simply not be, well, mathemtics! A similar prejudice exists throughout the university. I once knew a humanities professor who really believed that the quality of Truth is conferred on a paper by its being typeset and published in a prestigious journal. The idea that something True could exist unpublished in a drawer somewhere made no sense to her. Now mathematicians aren't that bad, of course, but their prejudice for the presentation

of mathematics in old-fashioned typefaces, with illustrations drawn by guys with green eye-shades, is strong and shows no signs of weakening.

Print is for the archiving of results. Print is for building careers. Print is for intimidating students. It is often not the best means for conveying ideas[1].

And this criticism does not only apply to mathematics. "It is a surprise to me that so little of the new physics is taught today in terms of pictures, particularly when it is being learned. Undoubtedly, much of the controversy dealing with quantum physics and its paradoxes arises from a paucity of pictures." — Wolf, Fred Alan, *Parallel Universes*, Simon and Schuster, N.Y., 1988, p. 114.

Cinematographic Presentation

"The view that change is constituted by a series of changing states [nineteenth century philosopher Henri Bergson] calls 'cinematographic'." — Russell, Bertrand, *A History of Western Philosophy*, Simon and Schuster, N.Y., 1972, p. 804. This form of presentation is what you see when a teacher or professor explains something at the blackboard. In the case of, say, a proof in geometry, the teacher typically begins by drawing a figure — a circle or triangle, say — and then, as he or she talks, he or she adds lines and labels. This type of presentation *never* appears in textbooks. A cynical person might say that's why students have to pay so much to go to college: because some things are simply not available anywhere else, e.g., in books. But there is absolutely no reason why this form of presentation can't be put into a book, although there is a trade-off: space for speed in understanding. Here is an example of such a presentation:

This type of presentation (in a book) can be improved if the important change in each successive drawing is highlighted, so that the eye can almost *see* how the argument goes, rather than the mind having to figure it out from a single drawing or, worse still, from text alone. Now in some cases, it is possible to number the parts of the figure in the order of the steps in the proof, and thus save all those drawings. In effect, what such numbers do is tell the reader how the drawing should be looked at. And why not go the next step, and with each number include the words constituting that step in the argument?

---

1. Computer graphics, in conjunction with the Internet, are now removing the last excuse of mathematicians for keeping illustrations to a minimum. The printed mathematical journal may continue to be the final source of prestige in the mathematical community, but more and more mathematical ideas (I don't say "papers" because the presentation is often much more informal than that of a paper) are being circulated and read and discussed and criticised via the net. The life of mathematics is moving away from the journals and into this new medium.

Pictures vs. Language to Talk About Pictures

In the section "Logical Structure" on page 64, we saw that there is a distinct difference between a picture (e.g., a map or an arrangement of squares, triangles, and circles) and the language which can be used to talk about the picture. This is a very important concept. Let us consider the latter example. If we knew that we were going to be presented with a series of such arrangements of figures, we might begin to collect language forms with which to describe such arrangements. Examples of these forms might be: "The ... is under the ... .", "The ... is to the left of the ... .", "The number of ...s is ... .", and various combinations of such forms, e.g., "The ... is under and to the left of the ... ." We would then have a language to describe any arrangement of geometric figures. We would not have to learn, or create, a new language for each figure that confronted us. An Environment is something like such a language.

Let us go to one extreme (because going to extremes is often instructive) and imagine trying to create a textbook (or Environment) with the *fewest words we could*. Imagine that a law had been passed requiring that math be taught only by mimes, or by animated cartoons in which the cartoon teacher could not speak, but only could express Strong Approval, Strong Disapproval, and something like Look Here! The teacher could, of course, use arrows, rectangles and other enclosing line drawings, and whatever symbols were essential. I hope you will agree that, at least in subjects with a geometric basis, e.g., calculus, complex number theory, and linear algebra, a great deal could be communicated this way. It would be a new way to learn math! It would (almost) be "math without words".

At the other extreme is picture-less mathematics, in which everything is expressed through prose, including, of course, equations, inequalities, and representations of statements in formal logic.

But it's never an either/or world. Sometimes, when a picture is too complicated to draw, you have to rely on logic — use it as your sole navigation system. Other times, a picture is much better than logic, at least to find the solution to the problem. Ultimately, of course, everything has to be presented in, or be reducible to, statements of logic.

The Promise of Computer Graphics

So far I have been speaking as though drawings and text are independent of each other, in the sense that changing one does not automatically cause a change in the other — I mean, a physical change on the page or on the computer screen. But the computer has already begun to make each form of representation dependent upon the other. There are a number of programs which generate curves and surfaces from

specifications typed in by the user. It is only a matter of time before the reverse is possible: given a curve, say, a parabola displayed on the screen, you will be able move it, rotate it, and observe how its equation changes. Similar programs might allow you to move a tangent line along a curve and observe how the numerical value of the slope changes, or allow you to begin with a chord intersecting the curve, with the value of the slope being displayed, and then shorten this chord and watch the value of the slope changes.

Assume You Will Forget!

Memorization seems always to have held a special place in the minds of educators. Socrates is reputed to have complained that the invention of writing would only curb man's intellectual progress, since the memorization of knowledge would no longer be necessary. Similar remarks have been made by educators regarding the use of calculators — that students will no longer know the rules for doing arithmetic — and I am certain they will be repeated in response to the idea of a new way of organizing mathematical knowledge which is aimed at relieving the student of as much memorization as possible. Which, of course, does not mean that an Environment makes it more difficult for you to memorize mathematical information! Quite the contrary. In fact, as you will find out, it makes memorization much easier than a traditional textbook does. All an Environment does is to relieve you of the *need* to memorize as much as you have in the past in order to work homework problems and open-book exam problems.

In my opinion, memorization should be viewed as a natural byproduct of repeated use. We should (and usually do) memorize the information we use most often, and look up in the Environment the information we use less often. I realize that many math professors think otherwise, but that doesn't necessarily mean they are right.

So, in building an Environment, you should always proceed as though you were building it *for next time* — for the next time you will have to use it to solve a problem — or, better still, proceed as though you were building it for another student. **Just like we program for the next programmer. You take notes for when you will have forgotten 4/23/02** If you build it solely for yourself to use this time, there is a great danger you will say to yourself, "Oh, I'll remember this; I understand it now, surely I'll understand it next time — even in the next course — even a year or two from now!" But chances are you won't. So why not do the job once, write it down, and save all that time in the future when you will have to go through at least part of the labor you have just gone through?

Computer programmers — the better ones — apply a similar idea, namely, that of programming for the next programmer, meaning that they make a point of adding

comments to their programs that they feel the next programmer would most want to see. The fact that the next programmer might well be themselves in no way invalidates the importance of the idea.

In building an Environment, you are *pre-learning*, *pre-processing* for the next time you use it.

This might be a good place to point out another fundamental difference between classroom teaching that is based on the logical structure of a subject and one based on problem-solving structure. Each implies a model of the typical student's learning behavior. The model implied by typical classroom teaching is that students, after being taught new knowledge, can, or should, retain it indefinitely into the future — not only until the next set of homework problems, or until the next exam, or until the finals, but indefinitely thereafter, as is clearly implied by the notion of prerequisites for college courses. Furthermore, the model implies that memorization is the best way for them to retain fast access to the knowledge whenever they need it.

Contrast this model with the one implied by the Environment approach. Here the model implies that students will probably not remember what they should, that they will frequently need to look things up (fast!), and that they will occasionally, perhaps often, have to take courses without having "taken" one or more prerequisites, so that they will need to look up material in the prerequisite courses fast.

Which model is the more realistic one, based on your own experience?

Finally, lest you think that a dislike for memorization is a sign of a loser, or at least of a second-rate intellect:

"Einstein duly graduated [from the Swiss Federal Polytechnic, one of the best schools of its kind in Europe] in 1900, but the four years of hard slog had taken their toll:

'One had to cram all this stuff into one's mind for the examinations, whether one liked it or not. This coercion had such a deterring effect on me that, after I had passed the final examination, I found the consideration of any scientific problems distasteful to me for an entire year.' — Einstein, Albert, quoted in Hollingdale, Stuart, *Makers of Mathematics*, Penguin Books, N.Y., 1989, p. 373.

Include Heuristics

Heuristics are procedures which get you the right answer most of the time. You may run into teachers (and possibly students) who say that "it shouldn't be necessary" to write down heuristics, that "if you understand the concepts, you won't need to write down the heuristics". These remarks might have some validity if every student were preparing for a life's work in the subject under discussion, but every student is not. Furthermore, the forcing yourself to write down the heuristic instead of forcing

yourself to somehow work it out in your mind and then somehow keep it straight in your mind, will enable you to start solving problems correctly sooner.

Of course, not all heuristics are algorithmic: not all heuristics are guaranteed to get you the right answer every time. In some subjects, especially at the start, the heuristic for doing proofs may amount to nothing more than:

"1. Pick strategy which seems most likely to bring results, e.g., the strategy of proving the contrapositive, or of using proof by contradiction, or of working backward, or of working forward from the given **if** clause.

"2. Use theorem lists to try to construct proof. If no success within reasonable time, pick another strategy and repeat step 2, etc."

Try To Capture Intuitions

Mathematical intuitions can be divided into two classes: those pertaining to concepts and those pertaining to problem-solving in the subject. Examples of the first class are:

- the familiar description of continuity as a function whose curve has "no gaps";
- the wheel-and-spokes model of congruence given in "A Number Theory Environment (partial)" on page 203;
- Bertrand Russell's illustration of the Axiom of Choice via the problem of selecting the left shoe from each pair of an infinite set of pairs of shoes, vs. the problem of selecting one sock from each pair in an infinite set of pairs of socks;
- the idea that in the Lebesgue integral we sum by first placing the elements to be summed into categories, then sum the elements in each category, whereas in the Riemann integral, we simply sum "from the first element to the last".

Even broader intuitions like the following are useful. To repeat an earlier quotation:

"When I think about mathematical ideas, I see the abtract notions in symbolic pictures. They are visual assemblages, for example, a schematized picture of actual sets of points in a plane. In reading a statement like 'an infinity of spheres or an infinity of sets', I imagine a picture with such almost real objects, getting smaller, vanishing on some horizon." — Ulam, S. M., *Adventures of a Mathematician*, Charles Scribners Sons, N.Y., 1976, p. 183

Problem-solving skills, along with self-teaching skills, are the most difficult for the Environment to capture. You should first of all include the suggestions of professors (*when you can obtain such suggestions!*), e.g., "If I have to solve a problem of this type, I generally begin by ...." Second, you should (always) try to reduce searching as much as possible, by, e.g., categorizing and cross-referencing your lists of theorems, as described under "Theorems on the Entity" on page 82.

Use "..." Instead of *x*, *y*, *f*, *g*, as Generic Symbols

Normally, you put non-alphabetical symbols immediately following the alphabetical section of the Environment. Sometimes, however, you may want to have a place in your Environment for information on non-alphabetical symbols, e.g., {, }, or [,], but at the same time indicate the context. So you need a symbol that is more general than the generic *x*, *y*, *f*, *g*. If you use these letters as generic symbols, you then have to remember which symbol you used so that you can find it in the alphabetical ordering in your Environment. A way of getting around this problem is simply to use the ellipsis, ..., instead of these symbols.

Appendix A — Linear Algebra and Tensor Calculus Are Just "Great Big Tables"!

You don't need to take a *course* in a subject that can be made virtually 100% look-up-able! Two examples of such a subject are linear algebra and tensor calculus, the first being relatively "easy", the second being very difficult (even for Einstein!).

And yet, each year students go deeper in debt so they can sit and listen to the phone book[1] being read to them for an entire semester — and take notes and memorize content that can and should be made available to them in easily look-up-able format i.e., as an Environment.

One of these days, the light will dawn in the mind of some exceptional academic mathematician and he or she will realize at the very least that there is money to be made in the publication of Environments for one or both of these subjects.

He or she will begin by listing all the technical terms, symbols and frequently-occurring expressions in the subject and will be amazed by how long that list is. Then will come, first and foremost, the writing of definitions for each term and symbol in the list, them the formatting of related content as described above in this chapter.

The mathematician will, I think, be unable to avoid recognizing how much of the content of both subjects can and should be given over to the computer: in linear algebra, that includes solving systems of simultaneous linear equations (for now, the written procedure must be included in the Environment) and of course computing determinants. Of course theorems will be collected under appropriate headings.

In the tensor calculus, all the tedious calculations of outer and inner products, and of finding derivatives, and of raising and lowering indices and of performing contractions, etc., and, of course, of expanding abbreviated expressions, can and should be handed over to the computer[2]. If this is not possible, then written procedures for the student to follow, must be in the Environment.

Students need to be reminded that the tensor calculus is, among other things, a tool that turned out to be useful for representing the facts of Einstein's general theory of relativity.

---

1. Since the traditional phone book is becoming obsolete, I should probably say, "dictionary" here.
2. I have been told that there are currently available mathematical program packages for computing some 250 tensor calculus functions associated with Einstein's general relativity..

(If I were writing a textbook on general relativity, I would state the most important facts *first*, then do top-down proofs of each. In other words, I would proceed on a just-in-time-learning basis as far as the tensor calculus was concerned.)

t

Appendix B — "There Are No Difficult Subjects, Only Incompetent Presentations of Subjects"

I am hoping that by the time you have read this far in the book, and you have perhaps built at least one partial-Environment, you will not find the title of this Appendix to be absurd. You might even agree with it.

What brought the words into my mind was reflections on the way the tensor calculus is presented in textbooks, vs. the way it would be presented in a complete Environment. (See previous Appendix.) The idea that this subject should be presented in the traditional linear format of the typical textbooks, is a prime example of the dimness, the fundamental-not-getting-it, of mathematicians when it comes to the presentation of mathematical subjects. I could almost say, it is a prime example of the determination of mathematicians to maintain their prestige by keeping things difficult.