



Introduction

1

How to Use This Book

This book describes in detail the TUTOR language, which is used by teachers to create lesson materials on the PLATO IV computer-based education system. Teachers use the TUTOR language to express to the PLATO computer how PLATO should interact with individual students. Students and teachers interact with PLATO through terminals each of which includes a plasma display panel screen and a typewriter keyboard, as shown. Using TUTOR, an author of a computer-based lesson can tell PLATO how to display text, line drawings, and animations on the student's screen. The author can ask PLATO to calculate for the student, to offer the student various sequencing options, and to analyze student responses.

The TUTOR language was originally created and developed for educational purposes. However, educational interactions are probably the most subtle and difficult of all the interactions a person might have with the author of materials presented through a computer. It is now clear that other kinds of interactions are also handled well by means of TUTOR, including recreation and communication. Nevertheless, for concreteness this book will concentrate on the instructional applications of TUTOR.

It is hoped that you have already studied the textbook “Introduction to TUTOR” by J. Ghesquiere, C. Davis, and C. Thompson, and the associated PLATO lessons. These materials are designed to teach you not only basic aspects of TUTOR but also how to create and test your own lessons on the PLATO system. The present book, “The TUTOR Language,” does not attempt to describe the latter aspects, such as how to insert or delete parts of your lesson and how to try out your new lesson. It does cover all aspects of the TUTOR *language*: that is, *what* statements to give PLATO but not *how* to type these statements into a permanent PLATO lesson space. By studying this book you could, in principle, write down on paper a lesson expressed in the TUTOR language, but when you go to a PLATO terminal to type in your new lesson, you may not know what buttons to push to get started. Also, this book discusses TUTOR in more detail than does “Introduction to TUTOR,” which makes “The TUTOR Language” less appropriate for your initial study.

It is also hoped that as you study this book you will try things out at a PLATO terminal. TUTOR is designed for interactive use, in which case an author writes a short segment of a lesson, tries it, and revises it on the basis of the trial. Normally, the sequence write, try, revise, and try again takes only a few minutes at a PLATO terminal. It is far better to create a lesson this way than to write out a complete lesson on paper, only to find upon testing that the overall structure is inappropriate.

It is also helpful to try the sample lesson fragments discussed in this book. It is literally impossible to describe fully in this book how the examples would appear on a PLATO terminal. The PLATO medium is far richer than the book medium. One striking example is the PLATO facility for making animations such as a car driving across the screen. As another example, you must experience it directly to appreciate how easy it is at a PLATO terminal to draw a picture on the screen (by moving a cursor and marking points), then let PLATO *automatically* create the corresponding TUTOR language statements which would produce that picture. PLATO actually writes a lesson segment for you!

This book is written in an informal style. Sometimes, when the context is appropriate, topics are introduced in a different chapter than would be required by strict adherence to a formal classification scheme. In these cases, the feature is at least mentioned in the other chapter, and the index at the end of the book provides an extensive cross-linkage. The order of presentation, emphasis, examples, and counter-examples are all based on extensive experience with the kinds of questions working authors tend to ask about TUTOR.

If you are a fairly new TUTOR author, read this book lightly to get acquainted with the many features TUTOR offers. Plan to return to the book from time to time as your own authoring activities lead you to seek detailed information and suggestions. Your initial light reading should help orient you to finding appropriate sections for later intensive study. After you feel you know TUTOR inside and out, read this book carefully one last time, looking particularly for links among diverse aspects of the language. This last reading will mean much more to you than the first!

If you are already an experienced TUTOR author, read this book carefully with two goals in mind: to spot features unused in your past work but of potential benefit, and to acquire a more detailed understanding of the structural aspects of the language, with particular emphasis on judging.

The remainder of this introductory chapter contains some interesting examples of existing PLATO lessons, a description of the PLATO keyboard including the use of the special function keys, and a review of the most basic aspects of TUTOR.

Sample PLATO Lessons

Figures 1-1 through 1-6 on the following pages give several examples of interesting PLATO lessons. All were written in the TUTOR language. They have been chosen to give you some idea of the broad range of TUTOR language capabilities. Each example is illustrated with a photograph of the student's screen at a significant or representative point in the lesson. (See the note at the bottom of pg. 7.)

The PLATO terminal's display screen consists of a plasma display panel which contains 512 horizontal electrodes and 512 vertical electrodes mounted on two flat plates of glass between which is neon gas. Any or all of the quarter-million (512×512) intersections of the horizontal and vertical electrodes can be made to glow as a small orange dot. (The word "plasma" is the scientific name for an ionized gas; the orange glow is emitted by ionized neon gas.) As can be seen in the sample photographs, the PLATO terminal can draw lines and circles on the plasma panel as well as display text using various alphabets. Both drawings and text are actually made up of many dots. TUTOR has many display features for writing or erasing text and drawings on the plasma panel.

The TUTOR Language

Type your question about the unknown and then press NEXT.

When you have identified the compound press BACK.

» does it dissolve in H₂O

It is slightly soluble in water.

SCORE = 1

For tables of data press DATA. To review press LAB.

For help press HELP.

Fig. 1-1. Dialog in which a chemistry student attempts to identify an unknown compound by asking experimental questions. (Stanley Smith)

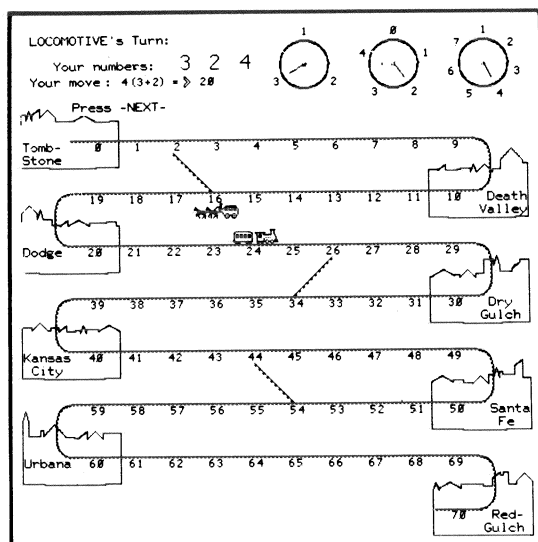


Fig. 1-2. Game of mathematical strategy in which two grade-school children compete in constructing advantageous mathematical expressions from random numbers appearing on the spinners. (Bonnie Anderson)

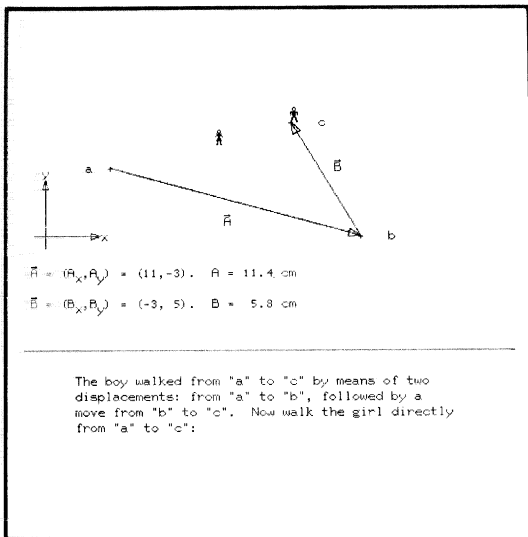


Fig. 1-3. Tutorial on vectors in which the student walks a boy and girl around the screen and measures their vector displacements. (Bruce Sherwood)

Translate:

Третий человек специалист по физике.
 The third man is a specialist in physics. ok

Девушка шла к маленькому музею.
 > The girl goes toward the small museum. no
 XXXX =====

The girl was going toward a small museum.

Fig. 1-4. Russian sentence drill. The markings under the student's translation of the second sentence indicate incorrect words and misspellings. (Constance Curtin)

The TUTOR Language

Add parts to the cell below to synthesize the protein chain...
Leucine--Aspartic acid--Glutamic acid

Δ-Leucine
o-Aspartic acid
α-Glutamic acid

GACTTAGTC
CTGATCAG

CUGAUCAG | | |
 GAC UUA GUC

What would you like to add?

(HELP, DATA, LAB, BACK)

Fig. 1-5a

DNA
GACTTAGTC

CUGAUCAG
m-RNA

OH, OH! You don't have all the parts!
In particular, ribosomes are missing

Fig. 1-5b

Graphical illustration of the biochemical steps involved in protein synthesis. The student introduces appropriate DNA, RNA, etc., into an initially empty cell, then watches the synthesis proceed. Here the synthesis breaks down for lack of a crucial part. (Paul Tenczar)

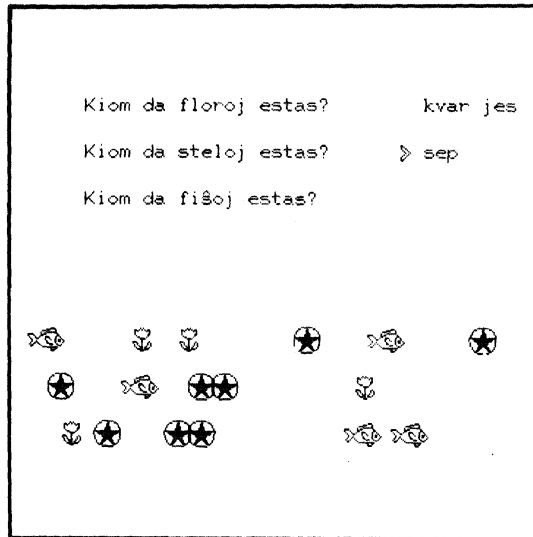


Fig. 1-6. Using graphics to teach Esperanto without using English. Here the stars have been circled to emphasize the student's mistake in counting. (Judith Sherwood)

These are actual photographs of the plasma panel. The display shows orange text and drawings on a black background, but the pictures are shown here as black on white for ease of reproduction. The plasma panel size is 22 cm. square (8.5 in. square).

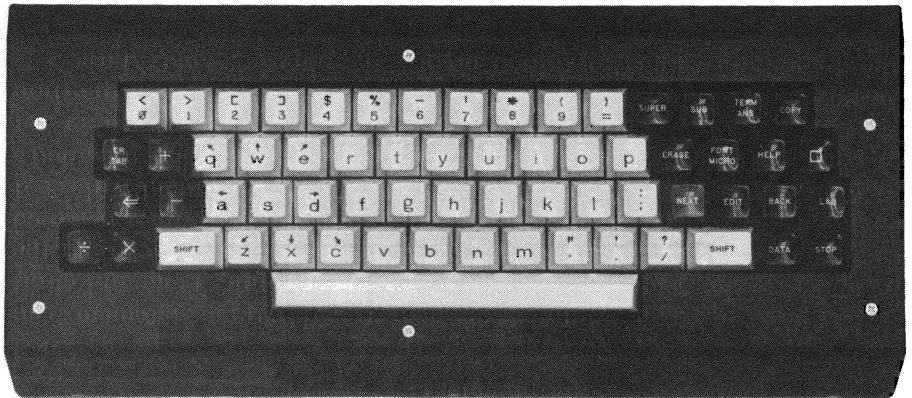


Fig. 1-7

The PLATO Keyboard

Every PLATO terminal has a keyboard like the one pictured above. The keyboard has a number of special features which are closely related to certain aspects of the TUTOR language, such as the HELP key which allows students to access optional sections of a lesson written in TUTOR.

The central white keys include letters, the numbers 0 through 9 along the top row, and punctuation marks. Note that the *numbers* 0 and 1 are different from the *letters* o and l. The zero has a slash through it to distinguish it unmistakably from the letter o. Except for these distinctions, the white keys are the same as the keys on a standard typewriter. Capital letters are typed by pressing either of the SHIFT keys while striking a letter key. Some keys show two different characters, such as the keys in the upper row, e.g., depressing a SHIFT key while striking a “4” produces a “\$”.*

Eight of the letter keys (d, e, w, q, a, z, x, and c all clustered around the s key) have arrows marked on them pointing in the eight compass directions. Typing “e” with a SHIFT key depressed normally produces a capital “E” on the screen, not a northeast arrow. The directional arrows are shown because these keys are sometimes used to control the motion of a cursor or pointer on the screen. In this context, the student presses an un-shifted “e” and the lesson interprets this as a command to move a cursor northeast on the screen, rather than a command to display an “e”

*Since this book deals with technical entities, which are set off by quotation marks, it is necessary to violate certain rules of punctuation.

on the screen. Such redefinitions of what a key should do in a particular context provide enormous flexibility. Another interesting example is the use of the keyboard to type Russian text in the Cyrillic alphabet.

Spaces (blank characters) are produced by striking the long "space bar" at the bottom of the keyboard. Holding down a shift key while hitting the space bar produces a backspace. An example of the backspace's use is in underlining. The underlined word "cat" is produced by typing "c", "a", "t", backspace, backspace, backspace, underline, underline, underline (underline is shift-6, not to be confused with the minus sign or dash). Typing "T", backspace, "H", will superimpose the two letters, making a "H." The backspace is used for superimposing characters, whereas the ERASE key (just to the right of the letter p) is used to correct typing errors.

A few black keys on the left side of the keyboard are mainly associated with mathematical operations: they include plus, minus (also used as a dash), times and divide (\div is equivalent to the slash /). The \Leftarrow is used in TUTOR calculations to assign values to variables. The TAB key is most often used by authors writing lessons rather than by students studying lessons. The TAB key's function is similar to the tabulate function on standard typewriters, e.g., pressing TAB once is equivalent to hitting the space bar as many times as is necessary to reach a preset column on the screen. Shift-TAB, called CR for "carriage return", to continue the typewriter analogy, moves typing down one line and to the left margin. Shift-plus produces a Σ (which means summation in mathematical notation) and shift-minus produces a Δ (which means difference in mathematical notation).

The black keys at the right of the keyboard are called "function" keys because they carry out various functions rather than displaying a character on the screen. By far the most important function key is NEXT. The cardinal rule for studying PLATO lessons is "When in doubt, press NEXT." Pressing NEXT causes the next logical thing to happen, such as proceeding on to a new display, asking for a response to be judged, erasing an entire incorrect response, etc. The second most important function key is ERASE, which is used to correct typing errors. Each press of ERASE erases one character from the screen. Pressing shift-ERASE (abbreviated as ERASE1) erases an entire word rather than a single character. Note the difference from the backspace (shift-space) which does not erase and is used for superimposing characters.

The EDIT key is also used for correcting typing. Suppose you have typed "the quik brown fox" when you notice the missing "c" in "quick". You could press ERASE1 twice to erase "fox" and "brown", use ERASE to get rid of the "k", then retype "ck brown fox". The EDIT key makes

such retyping unnecessary. Instead of hitting ERASE1, you press EDIT which makes the entire sentence disappear. Press EDIT again, and the entire first word "the" appears. Press EDIT again and you see "the quik" on the screen. Use ERASE to change this to "the quick". Now hit EDIT twice to bring in the words "brown" and "fox". The final result is "the quick brown fox". This takes longer to describe here in words, but pressing the EDIT key a few times is much easier and faster than doing all the retyping that would otherwise be necessary. The EDIT1 key (shift-EDIT) brings back the entire remaining portion of a sentence. For example, after inserting the "c" to make "the quick", you could hit EDIT1 once to bring back "brown fox". You should type some sentences at a PLATO terminal and study the effects produced by EDIT and EDIT1.

The COPY key is closely related to the EDIT key and is used mainly by authors. While EDIT and EDIT1 cycle through words you have just typed, COPY and COPY1 bring in words from a pre-defined "copy" sentence. These keys are used heavily when changing or inserting portions of a lesson.

The display "a^{2b}" can be made by hitting "a", then SUPER, then "2", then "b". SUPER makes a non-locking movement higher on the screen for typing superscripts. Notice that SUPER is struck and released, not held down while typing the superscript. Striking shift-SUPER makes a locking movement, so that the sequence "a", shift-SUPER, "2", "b" will produce "a^{2b}". The SUB key is similar to SUPER. For example, the display "H₂O" is made by typing "H", SUB, "2", "O". A locking subscript results from shift-SUB, which is also what is used to get down from a locking superscript. Similarly, shift-SUPER will move up from a locking subscript.

There are 34 additional characters not shown on the keyboard which are accessible through the MICRO key. For example, striking and releasing the MICRO key followed by hitting "p" produces a "π". The sequence MICRO-a produces an α. Typing "e", MICRO, "q", produces "è", whereas typing "E", MICRO, "q", produces "É". Note the "auto-backspacing" which not only backspaces to superimpose the accent mark but also places the accent mark higher on a capital letter. Six MICRO options involve autobackspacing: ` (q), '(e), "(u), ^(x), ~(n), and _(c). The last accent mark (MICRO-c) is used for creating cedillas (ç and Ç) and does not involve a different height for capitals. It is easy to remember these keys because of natural associations. The ` and ' accent marks are on the q and e keys which have the ↖ and ↗ arrows marked on them. The umlaut " usually appears on a "u" (German ü). The circumflex ^ is on the x key. The tilde ~ usually appears on an "n" (Spanish ñ).

The Greek letters α , β , δ , θ , λ , μ , π , ρ , σ , and ω are produced by typing MICRO followed by a, b, d, t, l, m, p, r, s, or w. Here is a complete list:

key	MICRO-key	key	MICRO-key
		\emptyset	\triangleleft
		1	\triangleright ("embed" symbols)
a	α (alpha)	< (shift- \emptyset)	\leq (less than or equal)
b	β (beta)	> (shift-1)	\geq (greater than or equal)
d	δ (delta)	[(shift-2)	{
t	θ (theta)] (shift-3)	} (braces)
l	λ (lambda)	\$ (shift-4)	# (pound sign)
m	μ (mu)	5	@ (each)
p	π (pi)	6	\triangleright (arrow)
r	ρ (rho)	=	\neq (not equal)
s	σ (sigma)) (shift=)	\equiv (identity)
w	ω (omega)	;	\sim (approximate)
q	` (grave)	o	$^\circ$ (degree sign)
e	˘ (acute)		(vertical line)
c	¸ (cedilla)	D	\rightarrow (east)
u	¨ (umlaut)	W	\uparrow (north)
n	~ (tilde)	A	\leftarrow (west)
x	^ (circumflex)	X	\downarrow (south)
C	© (copyright)	,	\updownarrow (special)
Q	(leftward writing)	+	& (ampersand)
R	(rightward writing)	/	\ (backwards slash)
CR (shift- TAB)	(special TAB for leftward writing)	x	o (matrix multiply)
		(shift-x)	\times (vector product)

These are the standard MICRO definitions. You can change these by setting up your own micro table. This is discussed in Chapter 9.

The standard character set includes all the characters we have seen so far, including the Greek letters and other characters accessible through the MICRO key. The shifted MICRO key, called FONT, lets you shift from this standard set of characters to another set of up to 126 special characters which you can design.

These special characters might be the Cyrillic, Arabic, or Hebrew alphabet, or they can be pieces of pictures, such as the characters \curvearrowright , \curvearrowleft , and \curvearrowright which form a car when displayed side by side: $\curvearrowright\curvearrowleft\curvearrowright$. Unlike MICRO which only affects the next keypress, FONT locks you in the

alternate “font” or character-set. You press FONT again to return to the standard font. The creation of new character sets is described in Chapter 9.

If the author activates it, the ANS key can be used by the student to get the correct answer to a question. This is discussed in Chapter 7. The shifted ANS key, TERM, when pressed causes the question “what term?” to appear at the bottom of the screen. At this point, you can type any one of various keywords in order to move to a different part of the lesson. The use of TERM is discussed in Chapter 5.

If you set up an optional help sequence, the student can press the HELP key to enter the sequence. The student can then press BACK (or BACK1) to go back to where he or she was when originally requesting help, or will be brought back to the original point upon completion of the help sequence. You could also specify a different help sequence accessible by pressing HELP1 (shift-HELP). The six keypresses HELP, HELP1, LAB, LAB1, DATA, and DATA1 can, if activated by the author, allow the student a choice of six different help sequences. You can also activate NEXT, NEXT1, BACK and BACK1, but these simply let the student move around in the lesson without remembering or returning to the original place. In other words, these four keys do not initiate help sequences. Usually, BACK is reserved for review sequences or similar situations where you want to back up.

The STOP key throws out output destined for the terminal. A useful example is the case of skimming through pages of text in an on-line catalog or collection of notes. If you decide you want to skip immediately to the next page, you might press STOP in order to avoid the wait required to finish plotting the present page.

The STOP1 or shift-STOP key plays a crucial role in PLATO usage. You press STOP1 to leave a lesson you are studying. When a student is ready to leave the terminal he or she presses STOP1, which performs a “sign-out” function. Among other things, the sign-out procedure brings the student’s permanent status record up to date so that days later he or she can sign-in and resume working at the same point in the lesson. When an author presses STOP1 to leave a lesson that he or she is testing, the author is taken back to a point in the PLATO system where he or she can make changes in the lesson before trying it again.

The key next to HELP, with the square (□) on it, is similar to the EDIT key, but retrieves one character at a time, instead of a whole word. It is particularly useful when used in association with the EDIT key. The shifted square key is presently used as the ACCESS key, as described in Chapter 9.

Basic Aspects of TUTOR

In their simplest form, lessons administered by the PLATO interactive educational system consist of a repeating sequence: a display on the student's screen followed by the student's response to this display. The display information may consist of sentences, line drawings, graphs, animations (moving displays)—nearly anything of a pictorial nature, and in any combination. The student responds to this display by pressing a single key (e.g., the HELP or NEXT key), by pointing at a particular area of the screen, by typing a word, sentence, or mathematical expression, or even by making a geometrical construction. Lesson authors provide enough details about the possible student responses so that PLATO can maintain a dialog with the student. The sequence of a display followed by a response is the basic building block of a lesson and is called a “unit” in the TUTOR language. This “display-response” terminology is convenient but is not intended to imply that the student is in a subservient position. Often what we will conventionally call the student “response,” is a question or a command issued to PLATO to respond with a display of some kind.

An author constructs a lesson by writing one unit at a time. For each unit, the author uses the TUTOR language to specify: (1) the display that will appear on the student's screen; (2) how PLATO is to handle student responses to this display; and (3) how the current unit connects to other units.

A statement written in the TUTOR language appears as follows:

write	How are you today?
⏟	⏟
command	tag

The first part of the statement (-write-) is called the “command,” while the remainder (How are you today?) is called the “tag.” Command names mnemonically represent PLATO functions. The tag gives additional specifications on how the function is to be carried out. In this case, the tag specifies what text is to be written on the screen.

The following is an entire unit written in TUTOR. Figure 1-8 shows what a student would see on the screen while working on the unit.

The TUTOR Language

unit geometry
at 1812
write What is this figure?
draw 510;1510;1540;510
arrow 2015
answer <it,is,a> (right,rt) triangle
write Exactly right!
wrong <it,is,a> square
write Count the sides!

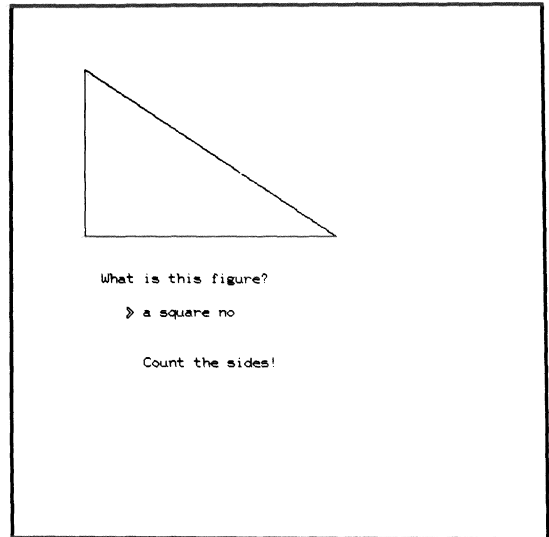


Fig. 1-8

We will discuss each statement of this unit in detail.

unit geometry

The -unit- statement initiates each unit. The tag (geometry) will become useful later when units are connected together to form a lesson. Each unit must have a name. No two units in a lesson may have the same name.

at 1812

The -at- statement specifies at what position on the screen a display will occur. The tag "1812" means that we will display something on the 18th line in the 12th character position. The top line of the screen is line 1 and the bottom line is line 32. There are 64 character positions going from 01 at the left edge of the screen to 64 at the right. Thus, 101 refers to line 1, character position 01 (the upper left corner of the screen), while 3264 refers to line 32, character position 64 (the lower right corner of the screen). Note that "0" means the number zero, as distinct from the letter "O".

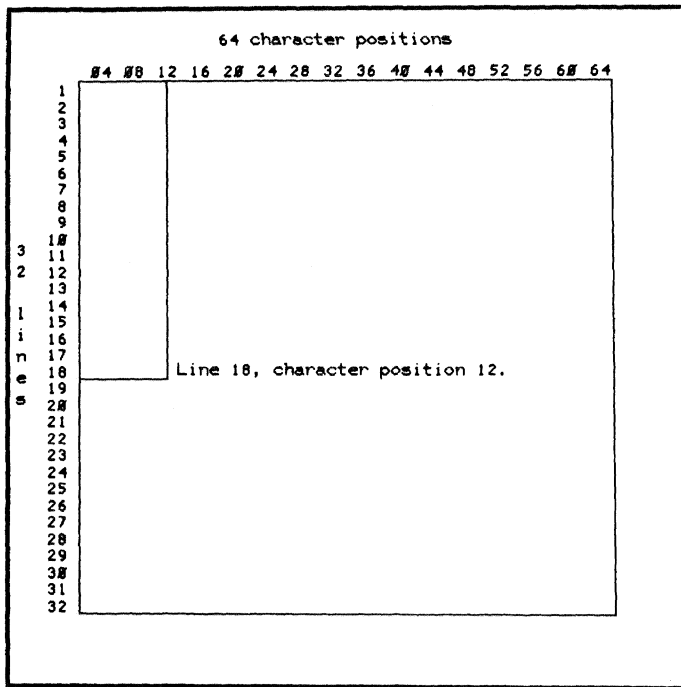


Fig. 1-9. Illustration of "at 1812"

`write What is this figure?`

The `-write-` statement causes the text contained in the tag to be displayed on the student's screen. The writing starts at line 18, character position 12, as specified by the preceding `-at-` statement.

`draw 510;1510;1540;510`

The `-draw-` statement specifies a straight-line figure to be displayed on the screen. In this particular case a series of straight lines will be drawn starting at location 510 (line 5, character position 10), going vertically downward to location 1510, then to the right to location 1540, and finally back to the starting point, 510. This produces a right triangle on the student's screen.

`arrow 2015`

The `-arrow-` statement acts as a boundary-line that separates preceding display statements from the following response-handling statements. Thus, what precedes the `-arrow-` command produces the screen display which remains visible while the student works on the question. Statements after the `-arrow-` command are used in handling student responses to the display. In addition, the `-arrow-` statement notifies TUTOR that a student response is required at this point in the lesson. The tag of the `-arrow-` statement locates the student response on the screen. An arrowhead is shown on the screen at this place to indicate to the student that a response is desired and to tell him or her where the response will appear. In this case the arrowhead will appear on line 20, character position 15. The student's typing will start at 2017, leaving a space between the arrowhead and the student's first letter.

```
answer <it,is,a> (right,rt) triangle
.
.
.
.
wrong <it,is,a>square
```

The `-answer-` and `-wrong-` statements are used to evaluate the student's response. The special brackets `<` and `>` enclose optional words, while the parentheses enclose important words which are to be considered synonyms. Thus any of the following student responses would match the `-answer-` statement: "a right triangle", "it is a rt triangle", "rt triangle", etc.

If the response matches the tag of the `-answer-` statement, TUTOR writes "ok" after the student's response. For a match to a `-wrong-` statement, "no" is written. An "ok" judgment allows the student to proceed to the next unit, whereas a "no" judgment requires the student to erase and try again. Any response not foreseen by `-answer-` or `-wrong-` statements is judged "no".

Having matched the student's response, TUTOR proceeds to execute any display statements following the matched `-answer-` or `-wrong-` statement. Thus, student responses of "a right triangle" and "square" will trigger appropriate replies. In the absence of specific `-at-` statements, TUTOR will display these replies three lines below the student's response on the screen.

Special help is provided to the student if his answer is partially

correct. Here is what happens if the student responds with “a lovely tringle, right?”:

> a lovely tringle, right?
 xxxxxxxxΔ----- ←

TUTOR automatically marks up the student’s response to give detailed information on what is wrong with the response. The word “lovely” does not belong here and is marked with xxxxxxxx, the Δ shows where a word is missing, the word “tringle” is misspelled and is underlined, and the word “right” is out of order, as is indicated by the small arrow.

Statements can be added to the current example unit which will greatly improve it. Consider the following:

	unit	geometry
	at	1812
	write	What is this figure?
	draw	510;1510;1540;510
	arrow	2015
	specs	okcap
	answer	<it,is,a> (right,rt) triangle
	write	Exactly right!
Handling additional responses	answer	<it,is,a> three*sided (right,rt) polygon
	write	Yes, or a right triangle.
	wrong	<it,is,a> triangle
	at	1605
	write	Please be more specific. It has a special angle.
	draw	1410;1412;1512
	wrong	<it,is,a> square
	write	Count the sides!

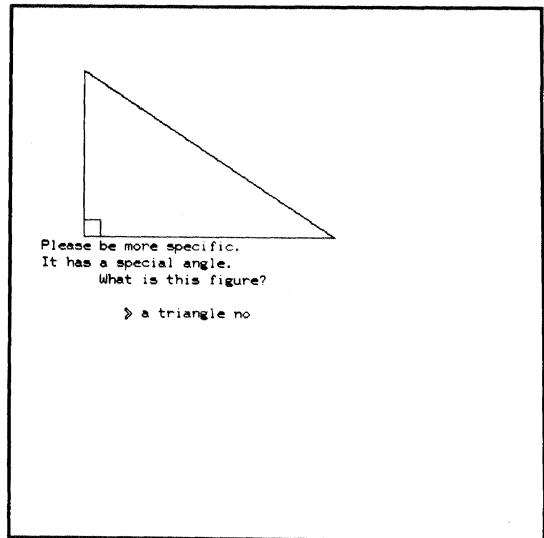




Fig. 1-10.

As you can see, any number of -answer- and -wrong- statements can be added to the response-handling section of the unit. Time and effort spent by an author in providing for student responses other than the common answer can greatly increase the ability to carry on a personal dialog with each student. Figure 1-10 shows what the student will see if he responds with “a triangle”. The construction “three*sided” is called a “phrase”. A phrase is a set of words to be considered together for purposes of spelling and word order. As another example, a question about Columbus’s flagship might involve the phrase “Santa*María”.

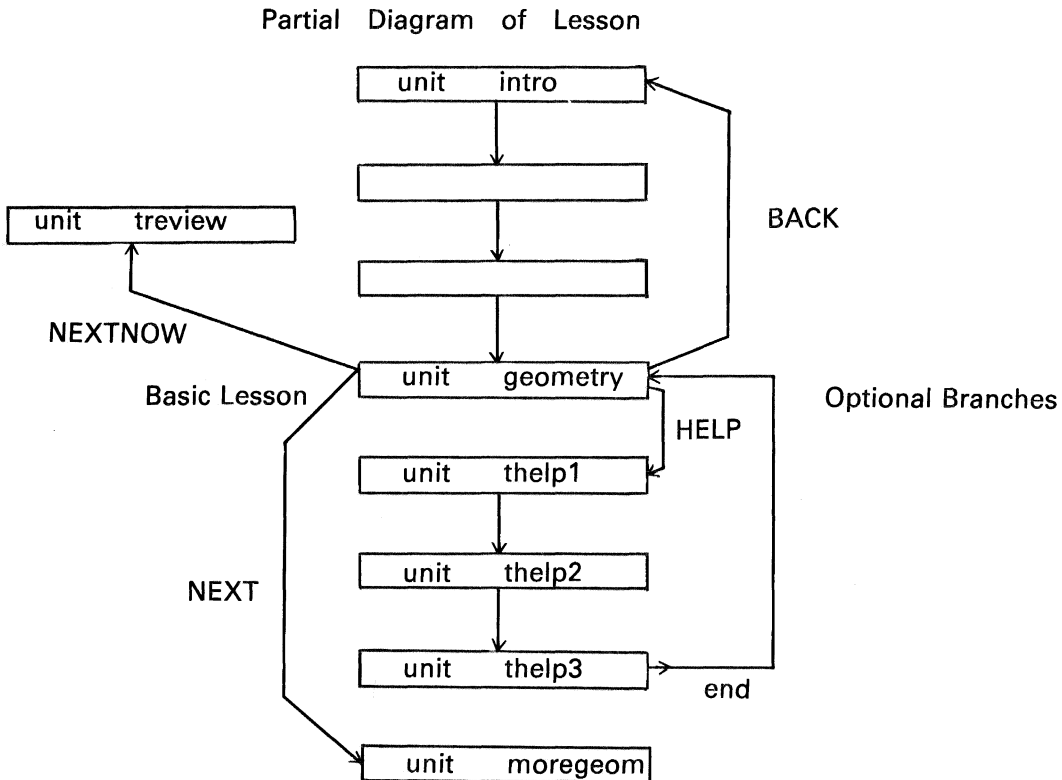
The `-specs-` statement is introduced here. It is used to give optional specifications on how the student's response is to be handled. In this case the tag, "okcap", specifies that any capitalization in the student's response is optional. Without this specification, TUTOR would consider "Right Triangle" to be misspelled. There are many convenient options available in a `-specs-` statement. For example, "specs okextra,noorder" specifies that extra words not mentioned explicitly in following `-answer-` and `-wrong-` statements are all right, and that the student's word order need not be the same as the word order of the `-answer-` and `-wrong-` statements to achieve a match. Such options can be used to greatly broaden the range of responses which can be handled properly.

Lessons could be written using only the commands already discussed. Explanatory units could be written using only display commands. Tutorial units could be interspersed to test a student's understanding of the lesson material. Thus a single linear chain of units could form a lesson. However, mastery of a few more TUTOR commands opens up a wealth of "branching" or sequencing possibilities. Branching, the technique of allowing alternate paths through a lesson, is one of the keys to personal dialog with each student. The example unit will, therefore, be expanded to include `-next-`, `-nextnow-`, `-back-`, and `-help-` commands:

	unit	geometry
	next	moregeom
	help	thelp1
	back	intro
	at	1812
	write	What is this figure?
	draw	510;1510;1540;510
	arrow	2015
	specs	okcap
	answer	<it,is,a> (right,rt) triangle
	write	Exactly right!
	wrong	<it,is,a> triangle
	at	1605
	write	Please be more specific. It has a special angle.
	draw	1410;1412;1512
	wrong	<it,is,a> square
	nextnow	treview

The tag of the `-next-` statement following the `-unit-` command gives

the name of the next unit the student will see upon the successful completion of unit “geometry”. The `-next-` statement is necessary because the next unit for a student in a highly-branching lesson sequence may not be the unit following in the lesson. For example, a diagram of the lesson flow involving unit “geometry” might be:



In moving from one unit to another the screen normally is automatically erased to make room for the displays produced by the following unit.

The `-help-` statement refers to a help unit which the student may reach through the use of the HELP key. Help units are constructed in the same manner as unit “geometry”. However, the last (or only) unit in a help sequence is terminated by an `-end-` command. Upon completing the last help unit, the student is returned to the “base” unit, the unit from which the student branched (in this case unit “geometry”). The student

need not complete the entire help sequence. He may press BACK or shift-BACK to return to the base unit from any point in the help sequence. Help units for unit “geometry” could appear as follows:

```
.  
.   
.   
* These units are help units for “geometry”.  
unit  thelp1  
at    1828  
write The figure has three sides.  
draw  510;1510;1540;510  
*  
unit  thelp2  
at    1828  
write It also has three angles.  
draw  510;1510;1540;510  
*  
unit  thelp3  
at    1828  
write Note the right angle.  
draw  510;1510;1540;510  
end
```



Any statement which begins with an asterisk (*) has no effect on the operation of the lesson and may be used anywhere to insert comments which describe the units. A comment statement between units improves readability by guiding the eye to the unit subdivisions of the lesson.


The -back- statement permits the student to move to a different unit by pressing the BACK key. Because of its name, it is customary to associate a review sequence with the BACK key. If a student is in a non-help unit that does not contain a -back- statement, the BACK key does nothing. In a help-sequence unit that has no -back- statement, the BACK key returns the student to the original base unit.

If the student calls the figure “a square”, he or she will see this response judged “no” and get the reply “Count the sides!” The -nextnow- statement is used to force the student through additional material. It locks the keyboard so that only the NEXT key has any effect. In particular, the student cannot erase his or her response. When the student presses NEXT, he or she will be sent to unit “treview”. Upon

completion of one or more units of review about triangles, the author might return the student to unit “geometry”. Thus, this student’s lesson flow might consist of:

- 1) a discussion of geometric figures;
- 2) a question about a right triangle;
- 3) an error causing -nextnow- to lock the keyboard;
- 4) further study of triangles; and finally
- 5) a return to the right triangle.

Consider now the problem of using unit “geometry” for a second student response. Additional display information is needed to ask the student a second question, and another -arrow- command is needed plus a second set of response-handling statements. The unit could appear as follows:

unit	geometry	
next	moregeom	
back	intro	
help	thelp1	
at	1812	
write	What is this figure?	
draw	510;1510;1540;510	
arrow	2015	
.	} Response-handling statements for first arrow.	
.		
.		
.		
 endarrow		
at	2512	
write	How many degrees in a right angle?	
help	angles	
arrow	2815	
.	} Response-handling statements for second arrow	
.		
.		
.		

The -endarrow- command delimits the response-handling statements associated with the first -arrow-. Only when the first -arrow- is satisfied by an “ok” judgment will TUTOR proceed past the -endarrow- command to present the second question. The statement “help angles” overrides the earlier statement “help thelp1”. If the student presses the HELP

The TUTOR Language

key while working on the second -arrow- he or she will reach unit "angles" rather than unit "thelp1".

The second question could have been given in a separate unit rather than following an -endarrow- command. The major difference is that the entire screen is normally erased as the student proceeds to a new unit, whereas here the second question was merely added to the existing screen display. Even if there is only one -arrow- command in a unit, -endarrow- can be useful, for it can be followed by display or other statements to be performed only after the -arrow- is satisfied. This is particularly convenient if there are several -answer- commands corresponding to several different classes of acceptable responses.

Fourteen TUTOR commands have been illustrated in this chapter. This repertoire is adequate to begin lesson writing. If you have access to a PLATO terminal, it would be useful at this point to try out the ideas discussed so far.