TUTOR User's Memo. Introduction to TUTOR.


Illinois State Office of the Superintendent of Public Instruction, Springfield.; National Science Foundation, Washington, D.C.

Mar 73

42p.

MF-$0.65 HC-$3.29

*Computer Assisted Instruction; *Computer Programs; Computers; *Manuals; Programing; *Programing Languages

This manual provides newcomers to the PLATO computer-assisted instructional system with an introduction to TUTOR, the system's language. It explains the basic commands and specific operating instructions and presents a set of seven simple exercises, the performance of which serves as an introduction to PLATO for the user. Four appendixes are also included: A) Summary of TUTOR Commands; B) How to Predict the Operation of a TUTOR Unit; C) Aids for Authors; and D) Using the TUTOR Editor. (PB)
TUTOR USER’S MEMO

INTRODUCTION TO TUTOR

March, 1973

Computer-based Education Research Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Computer-based Education Research Laboratory
University of Illinois Urbana Illinois
TUTOR USER'S MEMO

INTRODUCTION TO TUTOR

March, 1973

Computer-based Education Research Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
INTRODUCTION TO TUTOR

Writing on the Student’s Screen

TUTOR is the language in which instructional material is prepared for the PLATO Computer-based Education System. A lesson writer in TUTOR consists of a series of statements of the form,

COMMAND TAG

where "COMMAND" represents an operation to be performed by the computer and "TAG" represents specific instructions for performing the operation. For example, PLATO presents information to the student by means of a display screen. One TUTOR command which permits the author of lesson material to write on the student’s screen is the -write- command. The "tag" of the -write- command is simply the textual material which is to be displayed. Thus,

write How are you?

would display the message “How are you?” on the student’s screen. Since it is often necessary to be able to say exactly where material is to be displayed, the TUTOR command -at- is also available. The statements:

at 1020
write How are you?

specify that writing of the message "How are you?" is to begin on the 10th line of the student’s screen and at the 20th character position of that line (a PLATO screen holds 32 lines of up to 64 characters each). Once a starting position is specified for a display the -write- tag may include positioning of subsequent lines. Thus,

at 140
write PAGE 2
at 1510
write This is line 15,
This is line 16, and
This is line 17.
Using the TUTOR Editor

A description of the TUTOR editor is included as Appendix D of this document. Also, when you are using the editor, assistance is available via the -HELP- key. Here, we merely list the steps necessary before you start the exercise sets which begin on page 3.

a) Learn the name of the lesson space assigned to you.
b) Enter the editor (See sign-on procedure in Appendix D.)
c) Type your lesson name on the EDIT OPTION page and then press -NEXT-.
c') Specify descriptive information about your lesson. (See Appendix D for instruction.)
   This step is necessary only the first time you enter a new lesson. When you have finished entering the data requested, press -BACK-.
d) Press -a- on the LESSON EDIT page.
d') Choose a name for your block "a" (This step is done only the first time you enter any block).
e) Press -i- to begin inserting the text of exercise 1. (Be sure the words "INSERT MODE" appear at the top of your screen.)

The general pattern for entering each line of TUTOR text is as follows:

1) Type the command.
2) Press -TAB- key. (located on the left-hand side of your keyboard)
3) Type the tag.
4) Press -NEXT-.
5) Repeat steps (1) through (4) for the next line of your program.
See Appendix D for details on correcting mistakes and for the various special options available to you.

Exercise Set 1

1-1. Copy these statements into a block of your lesson.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>test</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>101</td>
<td>On line 1, position 1,</td>
</tr>
<tr>
<td>write</td>
<td>Upper left</td>
<td>write: &quot;Upper left&quot;</td>
</tr>
<tr>
<td>at</td>
<td>153</td>
<td>On line 1, position 53,</td>
</tr>
<tr>
<td>write</td>
<td>Upper right</td>
<td>write: &quot;Upper right&quot;</td>
</tr>
<tr>
<td>at</td>
<td>1601</td>
<td>On line 16, position 1,</td>
</tr>
<tr>
<td>write</td>
<td>Middle left</td>
<td>write: &quot;Middle left&quot;</td>
</tr>
<tr>
<td>at</td>
<td>3201</td>
<td>On line 32, position 1,</td>
</tr>
<tr>
<td>write</td>
<td>Lower left</td>
<td>write: &quot;Lower left&quot;</td>
</tr>
</tbody>
</table>

Read this material into the computer’s memory and look at it from a student’s point of view. (Press SHIFT-STOP)

1-2. Go back into Author Mode (from Student Mode, press SHIFT-STOP). Add to the program statements of exercise 1 to do the following:

a. In the center of the student’s screen write the letter ‘X’.

b. In the lower right corner of the student’s screen write the words:

   Lower right
   When finished, enter student mode again to check the placement of the various messages on the student’s screen.

Questions and Answers

In the previous exercises you learned a little about using TUTOR to write on a student’s display screen. Using the -at- and -write- commands, you can instruct the computer to present a question to the student. For example, the statements:

   at 503
   write Honolulu is in what state?

would present a question on line 5 of the screen starting at position 3. We must now learn to provide the student with a means of answering questions and to provide the computer with a
means of checking the answers given.

The presence of an activity requiring a student response is signaled by the statement:

\[ \text{arrow} \ 715 \]

where the tag (715) has exactly the same format and meaning as that of an -at- command. When the computer executes such a statement, it plots an arrow on the student's screen at the stated screen position. That is where the student's answer will appear when he types it. (In the above example, the student's answer will start on line 7, position 15.)

Once the presence of a response-requiring activity is signaled, we need a way to inform the computer of the response(s) to expect. The simplest such statement is one line:

\[ \text{answer} \ \text{Hawaii} \]

Such a statement informs the computer of a correct answer. When the student types a word that matches the tag of this statement, he is told that his answer is correct. (Even if a student misspells 'Hawaii' it will generally be recognized and underlined as a misspelling.)

In summary, a simple question-answer situation is written (in TUTOR) like this:

\[ \text{at} \ 503 \ 
\text{write} \ \text{Honolulu is in what state?} \ 
\text{arrow} \ 715 \ 
\text{answer} \ \text{Hawaii} \]

On line 5, position 3 write the question. Student's answer to be written on line 7 starting at position 15. A correct answer.
Exercise Set 2

2-1. Copy this program.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAC</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>math</td>
<td>On line 2, position 5, write the directions;</td>
</tr>
<tr>
<td>at</td>
<td>205</td>
<td>skip a line;</td>
</tr>
<tr>
<td>write</td>
<td></td>
<td>write an addition problem, skip a line;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>write a multiplication problem.</td>
</tr>
<tr>
<td></td>
<td>3 + 3 =</td>
<td>Answer to the addition problem is 6, and goes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>on line 4, position 13.</td>
</tr>
<tr>
<td></td>
<td>4 x 3 =</td>
<td>Answer to the multiplication problem is 12,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and goes on line 13, position 9.</td>
</tr>
<tr>
<td>arrow</td>
<td>413</td>
<td></td>
</tr>
<tr>
<td>answer</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>arrow</td>
<td>613</td>
<td></td>
</tr>
<tr>
<td>answer</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

Note: We intend that you skip a line between the instructions to the students and the first problem. Also that you skip a line between problems. You should not leave other blank lines in your program. Our samples contain blank lines so that we can fit in the explanations on the right. Do not copy those blank lines.

Enter student mode (press SHIFT-STOP) and go through it as a student. Try various combinations of correct and wrong answers.
2-2. One frequently wishes to give replies to a student contingent upon specific correct or wrong answers. This exercise illustrates such a procedure. Add only the bracketed lines to the unit from Exercise 2-1. (The statements from Exercise 2-1 are in italics.)

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>math</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>Answer these problems.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 + 3 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 x 3 =</td>
<td></td>
</tr>
<tr>
<td>arrow</td>
<td>413</td>
<td></td>
</tr>
<tr>
<td>answer</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>420</td>
<td>Insert this answer contingent writing. When student answers correctly, congratulate him, putting the message on line 4, position 20.</td>
</tr>
<tr>
<td>write</td>
<td>Very good</td>
<td></td>
</tr>
<tr>
<td></td>
<td>613</td>
<td></td>
</tr>
<tr>
<td>answer</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>wrong</td>
<td>7</td>
<td>Insert this wrong answer contingent writing for the multiplication problem. If the student accidentally adds, tell him so; starting the message at line 6, position 20.</td>
</tr>
<tr>
<td>at</td>
<td>620</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>Multiply, do not add.</td>
<td></td>
</tr>
</tbody>
</table>

Now read in (SHIFT-STOP) and go through the exercises as a student. Pay particular attention to the way the contingent lines of programming work. Be sure on one of your trials to answer the multiplication question (4 x 3 = ) with a '7'. 
2-3. On occasion, it is desired to have a message show only for one question before the answer is given. This is achieved by arrow contingent writing. To the material of Exercise 2-2, make these additions:

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>math</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>Answer these problems</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 + 3 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 x 3 =</td>
<td></td>
</tr>
<tr>
<td>arrow</td>
<td>413</td>
<td>Arrow contingent writing placed on line 7, position 1.</td>
</tr>
<tr>
<td>[at</td>
<td>701</td>
<td>Take you time</td>
</tr>
<tr>
<td>write</td>
<td>420</td>
<td>Very good</td>
</tr>
<tr>
<td>answer</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>420</td>
<td>A universal wrong contingency. No matter what wrong answer is given, say, &quot;Try again.&quot; On line 4, position 20.</td>
</tr>
<tr>
<td>write</td>
<td>Try again.</td>
<td></td>
</tr>
<tr>
<td>arrow</td>
<td>613</td>
<td>Arrow contingent writing for the multiplication question.</td>
</tr>
<tr>
<td>[at</td>
<td>801</td>
<td>Keep calm.</td>
</tr>
<tr>
<td>write</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>answer</td>
<td>7</td>
<td>Multiply, do not add.</td>
</tr>
<tr>
<td>wrong</td>
<td>620</td>
<td></td>
</tr>
</tbody>
</table>

Read in the lesson. (SHIFT-STOP). Observe the two examples of arrow contingent writing as well as the wrong answer contingent writing.
Separating Groups of Operations

In order to separate groups of operations which are to be performed together, the TUTOR command -unit- is used. The sequence:

```plaintext
unit one
at 201
write Page 1
unit two
at 201
write Page 2
```

will first display the words "Page 1" starting on line 2, position 1. When the student presses the key labeled "NEXT" the screen will be erased and the material in unit two will be shown. The words "Page 2" will appear. The tags of -unit- commands provide names for groups of TUTOR statements. Hence all of the TUTOR statements between the statement "unit one" and the statement "unit two" are referred to collectively as the "contents of unit one" or simply "unit one." Similarly all TUTOR statements between the statement "unit two" and the next following -unit- command are referred to as "the contents of unit two" or simply "unit two."
Exercise Set 3

3-1. Write a new unit that has the following problems on the screen:

\[ 275 \times 32 = \]
\[ 15 \times 5 = \]

a. Place this unit in the block with the material from Exercise Set 2 (just completed). Immediately after the last statement of the previous material, insert the statement:

```
unit arith2
```

b. As an arrow contingency for problem 1, write the message:

You may use pencil and paper.

c. As an arrow contingency for problem 2, write:

Do this one in your head.

d. As a universal wrong (\(-no\) command) for problem 1, write:

You had better use pencil and paper.

Place it near the bottom of the screen.

e. Make up one other answer-type contingency for one of the problems and include it in your program.
Sentence Answers

Many questioning situations require more than a single word response. For example, consider an exercise such as:

What type of figure is this?

There are several correct responses, including:

- It is a right triangle.
- a rt. triangle.
- a right triangular figure

To instruct the computer to accept answers of this type, we build a sentence checker using three types of words:

1. Words that MUST be present to be correct.
2. Words that can safely be IGNORED.
3. Words that CAN'T be present to be correct.

Here is a statement that will correctly check answers to the triangle question:

answer  <It, is, a, the, it's figure, polygon> (right, rt)
(triangle, triangular)

The first group of words (enclosed by < > ) consists of those words that may be ignored. Following this are two groups of words enclosed by parentheses. These are the MUST words. Each set of parentheses encloses a single must-type word together with acceptable synonyms. Notice that synonyms are separated by commas. If no synonyms are listed for a MUST word, parentheses are not used.
Exercise Set 4

4-1. Here is the complete unit described above.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>triang</td>
<td></td>
</tr>
<tr>
<td>figure</td>
<td>310,810,825,310</td>
<td>Draw the triangle, one segment from 310 (line 3, space 10) to 810 (line 8, space 10), a second segment from 810 to 825, and a third segment from 825 to 310.</td>
</tr>
</tbody>
</table>

at 1105
write What type figure is this? The question.
arrow 1305
answer <it,is,a,it's,figure, polygon (right,rt) (triangle, triangular) Start of answer.
wrong <it,is,a> square If the student's answer includes the word 'square', give him this message.
write A square has four sides.

a. Copy this unit. For ease of viewing as a student, put it ahead of the units containing material from the previous exercise sets.
b. Read your lesson into the computer memory. (SHIFT-STOP)
c. Try the following answer variations, noting the computer's response each time:
   i. it is a triangle, right? (to see response to incorrect word order)
   ii. it is a right triangle. (to test spelling)
   iii. it is a squair. (to test spelling on a wrong)
   iv. it is a beautiful right triangle. (to see response to extraneous words)
4-2. Write a new unit that asks the questions:

What explorer discovered America in 1492?
What ships were used in his voyage?
You design the answer provisions for each question.

Sequencing Units

The tags of -unit- commands may also serve as "addresses."
If the author wants his units to be seen by the student in an
order other than the sequence in which they are written, he may
alter this sequence by using -next- commands. -next- uses the
name of a unit as its tag. Here is an example of the use of
-next-.

```
unit one
next two
write Page 1
unit hidden
write This message will not be seen by the student
unit two
write Page 2
```

The student first sees the words "Page 1" (on line 1, beginning
at postion 101 since no -at- command is given). When the student
presses -NEXT-, the screen is erased and the words "Page 2"
appear. In this example, the unit named "hidden" is never seen
by the student.

The -next- command can also be used to create repetitive
type exercises as is illustrated in Exercise Set 6.

Student Initiated Branches

The -HELP-, -LAB-, and -DATA- keys and the shifted-HELP,
shifted-LAB, and shifted-DATA keys do not function for students
unless they have been enabled by the lesson author. By inserting
the commands -help-, -lab-, -data-, or -helpl, -labl-, datal-,
an author enables the corresponding key. The suffix "l" refers
to the shifted keys.

The following units show a use of the -HELP- key. Any of
the other five keys could have been used in precisely the same
manner.
Exercise Set 5

5-1. Copy these units into your lesson space.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>jef</td>
<td></td>
</tr>
<tr>
<td>next</td>
<td>mad</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>810</td>
<td>unit mad will follow when unit jef is completed.</td>
</tr>
<tr>
<td>write</td>
<td>Who was the THIRD President of the United States?</td>
<td></td>
</tr>
<tr>
<td>help</td>
<td>hint1</td>
<td>enable the -HELP- key. If a student presses -HELP-, show him unit hint1.</td>
</tr>
<tr>
<td>arrow</td>
<td>1210</td>
<td></td>
</tr>
<tr>
<td>answer</td>
<td>&lt;T, Thomas, Tom&gt; Jefferson</td>
<td>These help units merely give hints.</td>
</tr>
<tr>
<td>unit</td>
<td>hint1</td>
<td>They might have asked questions too.</td>
</tr>
<tr>
<td>at</td>
<td>410</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>He was a Virginian.</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>1810</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>Press -NEXT- for another hint.</td>
<td>skip a line</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Press -BACK- if you would like to answer the question.</td>
</tr>
<tr>
<td>unit</td>
<td>hint2</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>He was also the chief author of our Declaration of Independence.</td>
<td>an -end- command signals the end of a sequence of help units.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(NOTE: The program for Exercise 5-1 continues on the next page.)
Try these units as a student. Go through them at least twice, once without pushing -HELP-, and once with help.

5-2. Write one or more help units for unit mad. Place them after unit hint2. Check them in student mode.

Student Variables

Each terminal has reserved for it a set of 150 computer words called Student Variables. These variables are numbered from 1 through 150. Each such variable can be thought of as an individualized storage location for various types of information.

Probably the most important use of the student variables is personalized calculation. A simple example of this is the task of keeping records on the number of problems worked and the number of errors made by each student. For such an application we interpret a variable as a storage location for a number. When referring to, say, variable 5, and wanting it interpreted as a rational number in decimal notation, we call it:

\[ v5 \]

Later you will learn to use these same 150 student variables as storage locations for alphabetic strings, or as storage locations for integers.
Exercise Set 6

6-1. Here is a unit that randomly selects one digit addends for a continuous addition drill. An explanation of the lines is at the right.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>adrill</td>
<td>Randomly select two numbers between and including 1 and 9; put the first number in student variable v1, the other number in v2.</td>
</tr>
<tr>
<td>randu</td>
<td>vl,9</td>
<td>Calculate correct answer; put it in student variable v3.</td>
</tr>
<tr>
<td>randu</td>
<td>v2,9</td>
<td>On 4th line, 10th space—show first addend; only 1 digit—then show other addend, allowing 2 spaces, one to separate the + and the numeral—then write an equal sign with a space before it. At this stage the student sees something like: 7 + 8 =</td>
</tr>
<tr>
<td>calc</td>
<td>v3 ← vl + v2</td>
<td>Include -arrow- to indicate that a response is required and to separate display commands from response command.</td>
</tr>
<tr>
<td>at</td>
<td>410</td>
<td>Check for correct numerical answer, allowing no difference from contents of v3 (the actual answer).</td>
</tr>
<tr>
<td>show</td>
<td>vl,1</td>
<td>Repeat the unit.</td>
</tr>
<tr>
<td>write</td>
<td>+</td>
<td>Repeat the unit.</td>
</tr>
<tr>
<td>show</td>
<td>v2,2</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>=</td>
<td>Repeat the unit.</td>
</tr>
<tr>
<td>arrow</td>
<td>422</td>
<td></td>
</tr>
<tr>
<td>ansv</td>
<td>v3</td>
<td>Repeat the unit.</td>
</tr>
<tr>
<td>next</td>
<td>adrill</td>
<td></td>
</tr>
</tbody>
</table>

a. Read in the unit. Work a few problems as a student, noting what happens on wrong answers.

b. To exit this continuous drill, press SHIFT-STOP.
6-2. Unit adrill is not very flexible. It gives the student no information on how close he is to being correct. Let's fix that.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>adrill</td>
<td></td>
</tr>
<tr>
<td>randu</td>
<td>v1,9</td>
<td></td>
</tr>
<tr>
<td>randu</td>
<td>v2,9</td>
<td></td>
</tr>
<tr>
<td>calc</td>
<td>v3 ← v1 + v2</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td>410</td>
<td></td>
</tr>
<tr>
<td>show</td>
<td>v1,1</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>+</td>
<td>as in Exercise 1</td>
</tr>
<tr>
<td>show</td>
<td>v2,2</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>arrow</td>
<td>42i</td>
<td></td>
</tr>
<tr>
<td>ansv</td>
<td>v3</td>
<td></td>
</tr>
<tr>
<td>next</td>
<td>adrill</td>
<td></td>
</tr>
<tr>
<td>wrongv</td>
<td>v3,1</td>
<td>See if student's answer is within 1 of the correct answer (v3)--- if so, tell him---</td>
</tr>
<tr>
<td>write</td>
<td>You're off by 1.</td>
<td>check for wrong arithmetic operation.</td>
</tr>
<tr>
<td>wrongv</td>
<td>v1 x v2</td>
<td>tell student.</td>
</tr>
<tr>
<td>write</td>
<td>You are multiplying!</td>
<td>For any other incorrect answer--</td>
</tr>
<tr>
<td>no</td>
<td></td>
<td>tell him.</td>
</tr>
<tr>
<td>write</td>
<td>you're way off.</td>
<td></td>
</tr>
</tbody>
</table>

Try several problems as student to check the wrong-answer contingent messages.
The author of a lesson which uses lots of variable can easily become confused about which variables he is using for each purpose. One aid to remembering the assignment of variables is to give them mnemonic names.

The `define` command associates such names with variables (or functions of variables). Below we have rewritten unit adrill with defined names in place of v1 and v2. Change your copy of adrill to match ours.

```
unit      adrill
define    mynames
  first=v1, second =v2
  result=v3
randu     first,9
randu     second,9
calc      result+first + second
.ar        410
.show      first,1
.write      +
.show      second,2
.write      =
.arrow     422
.ansv      result
.next      adrill
wrongv     result,1
.write      You're off by 1.
wrongv     first x second
.write      You are multiplying!
.no
.write      You're way off.
```

A lesson may contain several sets of defines; therefore each set is given a name. (We've used the name "mynames.") Following the set name are mnemonic names and the variables to which they shall refer.

Try the new unit adrill as a student. It should function precisely as did the first version.
6-3. The unit in Exercise 6-2 goes on forever. Let's change it so that after we have worked four problems correctly in a row, we end the drill and tell the student how many errors he made. To do this, we need a counter (let's define worked=v10) for the number of problems worked correctly and a counter for the number of errors (use errors=v11). However these counters must be set to zero before we begin the drill. This requires a new unit before adrill. Also, since the define command must precede any use of the defined names, we must move the define command into this new unit.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>initial</td>
<td></td>
</tr>
<tr>
<td>define</td>
<td>mynames</td>
<td></td>
</tr>
<tr>
<td></td>
<td>first=v1,second=v2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>result=v3</td>
<td>Set both counter to zero...</td>
</tr>
<tr>
<td></td>
<td>worked=v10,errors=v11</td>
<td></td>
</tr>
<tr>
<td>calc</td>
<td>worked ← errors ← 0</td>
<td>jump immediately to unit adrill; do not wait for student to press -NEXT-. The student is never aware of being in unit initial.</td>
</tr>
<tr>
<td>jump</td>
<td>adrill</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>adrill</td>
<td></td>
</tr>
<tr>
<td>randu</td>
<td>first,9</td>
<td></td>
</tr>
<tr>
<td>randu</td>
<td>second,9</td>
<td></td>
</tr>
<tr>
<td>calc</td>
<td>result ← first + second</td>
<td></td>
</tr>
<tr>
<td></td>
<td>worked ← worked + 1</td>
<td>Increment problem counter every time we choose a new problem.</td>
</tr>
<tr>
<td>at</td>
<td>410</td>
<td></td>
</tr>
<tr>
<td>show</td>
<td>first,1</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>show</td>
<td>second,2</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>
Read in and check all contingencies, including the score keeping. (Be sure to make some errors.)
6-4. Make a drill lesson with the following characteristics. Put it in a new unit ahead of all previous units.
   a. A multiplication drill with the first factor a two digit number (i.e., between 10 and 99 inclusive) and the second factor a one digit number.
   b. Write the problems in horizontal format (i.e., 27 x 3 =).
   c. Count the problems worked and end the drill after the 8th problem.
   d. Provide the following information to the student on wrong answers:
      i. If within 1 of the correct answer, tell him so.
      ii. If within 10% of the correct answer, say, "You're close." (Use the statement wrongv ____ 10% for this, filling the blank with the name of the variable you are using.)
      iii. If the student accidentally adds, tell him he has added, and count it as an error.
      iv. On all other errors say, "Try again."
   e. When the drill is finished report the number of errors to the student.

Try the lesson as a student to see that it works as described.

Joined Units

One frequently writes material that is used in several units. Such material might be a graph or other display that is repeated, or it might be a large set of answers used for several questions.
Of course, the material could be repeated at the point of each use. This is usually wasteful of both computer space and author time. A more efficient way is to put the material to be repeated in an out-of-the-way unit. The join command, using the name of the out-of-the-way unit as tag, causes the computer to act as though the material were inserted at the appropriate points.

The following example illustrates some uses of the join command.

```plaintext
unit boxes
define place, origin=vl
calc origin ← 915
join box Draw a box at 915.
calc origin ← 2030
join box Draw a box at 2030.
unit box
line origin, origin + 200
line origin + 200, origin + 204
line origin + 204, origin + 4
line origin + 4, origin
```

In Exercise 6-3 (the addition drill) it would have been convenient to use a joined unit instead of thrice repeating the lines:

```plaintext
calc errors ← errors + 1
worked ← Ø
```

Each occurrence of those two lines could have been replaced by

```plaintext
join count
```

where "count" is the name of a unit included somewhere in the lesson having the form:

```plaintext
unit count
calc errors ← errors + 1
worked ← Ø
```
Exercise Set 7

7-1. This exercise illustrates a way to have a set of several exercises executed in a different order by each student, or by a given student going through them several times. Put this in a new unit ahead of all previous units.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TAG</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>start</td>
<td></td>
</tr>
<tr>
<td>define</td>
<td>medley,choice=v10</td>
<td>Add defined names here as you find you need them.</td>
</tr>
<tr>
<td>at</td>
<td>402</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>Press next to begin the drill</td>
<td>An entry unit---</td>
</tr>
<tr>
<td>setperm</td>
<td>6</td>
<td>constructs a set of integers 1 through 6 from which draws will be made later.</td>
</tr>
<tr>
<td>unit</td>
<td>branch</td>
<td></td>
</tr>
<tr>
<td>randp</td>
<td>choice</td>
<td>Randomly select one of the integers provided by the -setperm- command (without replacement) and put it in variable choice. When all choices are used put zero in choice.</td>
</tr>
<tr>
<td>jump</td>
<td>choice,done,done, b1,b2,b3,b4,b5,b6</td>
<td>Go immediately to the exercise indicated by the value in choice. Do not wait for the student to press NEXT.</td>
</tr>
</tbody>
</table>

a. UNIT B1 is to be the following exercise, written as shown, near the middle of the screen:

\[
56 \\
+ 29
\]

This unit should contain the line:

next branch

somewhere before the -arrow- command. Put in whatever answer contingencies you wish.

b. UNIT B2 is to ask the student to name the 49th and 50th states.

(Include the line: next branch in this unit also).
c. Units B3 through B6 should be exercises of your design. (Include: next branch in each.) Try to include at least one sentence judging situation.

d. Finally, include the following:

```
unit done
at 503
write You have finished the lesson.
end lesson
```

Test these six exercises.

7-2. You have now completed 7 exercise sets. In a new unit ahead of the previous ones, build an index which shows the student a numbered list of descriptions of the material in the various exercise sets. Tell the student to press the number of the exercise he wishes to see. Correct answers consist of the numbers, each followed by a -jump- to the beginning unit of that exercise. Include in your index unit the line:

```
term index
```

This will permit the student to return at any time to the index for another choice by pressing the -TERM- key, typing the word index, and pressing -NEXT-.
APPENDICES

APPENDIX A.................Summary of TUTOR Commands

APPENDIX B.................How to Predict the Operation of a TUTOR Unit

APPENDIX C................Aids for Authors

APPENDIX D.................Using the TUTOR Editor
APPENDIX A

Summary of TUTOR Commands

To be useful this summary must be brief. For more information on the commands (exceptions, special uses, etc.) see the various author aids that are available. (See Appendix C.) The classification scheme in this list is identical to that in lessons aid2 and aid3.

PRESENTING

Screen text

at specifies starting position of display on plasma panel
write displays message on the plasma panel
writec displays one of a series of messages contingent on the value of an expression
show displays the value of a variable (base 10) in the specified format
showo displays the value of a variable in octal notation (base 8)
showa displays stored or packed characters from variable(s) specified in tag
erase erases screen, selectively or entirely
mode specifies terminal writing state
inhibit temporarily disables certain normal actions of TUTOR
size changes to line-drawn characters larger than normal characters, size specified by tag
rotate causes line-drawn characters to be written at an angle specified in the tag
delay permits specification of short output delays
char permits specification of specially designed characters for display
plot displays special characters specified by -char- command
codeout specifies code sent to terminal to perform special display functions
Screen graphics

line
- draws line on screen between locations specified in tag

liner
- draws line from last writing location to location specified in tag

figure
- draws connected lines from tag1 to tag2 to tag3... to tag50 locations (coarse grid)

figuref
- draws connected lines from tag1, tag2 to tag3, tag4... to tag49, tag50 (fine grid)

circle
- draws circle with relevant parameters set in tag

circleb
- draws broken circle

dot
- plots single dot on screen at tag location

window
- delimits area on line-drawn display on plasma panel

Grafpack commands

origin
- specifies position for origin; all other plotting and writing with grafpack commands are relative to origin

axes
- specifies and draws axes

bounds
- establishes boundaries on axes but does not draw axes

frame
- draws frame around boundaries specified in tag or by previous -axes- or -bounds- commands

scalex
- specifies maximum (and minimum if given) values on axes

scaley
- (all subsequent commands are in scaled units)

locate
- similar to -at- but relative to origin and in scaled units

graph
- places dot or other specified character string at position indicated relative to origin and in scaled units

gline
- like -line-, -liner- commands except relative to origin and in scaled units

gliner
- like origin and in scaled units

labelx
- labels x axis (numbers and tick marks)

markx
- tick marks only

labely
- labels y axis (numbers and tick marks)

marky
- tick marks only

vbar
- draws vertical bar relative to origin, in scaled units

hbar
- draws horizontal bar relative to origin, in scaled unit
Non-screen

slide turns on slide projector and selects slide specified in tag
audio plays audio message specified in tag
play plays back audio recording, with message location specified in tag
record records message at location specified in tag
enable allows inputs from external device (e.g., touch panel)
disable refuses inputs from external device (e.g., touch panel)
ext sends value of tag variable to external accessory

Special typing controls

tabset sets tabs which can be used by student pressing -TAB- key
copy specifies string which is to be copied when -COPY- (carat) key is pressed
micro specifies micro table definitions used when the student presses -MICRO- key and then another specified key
charset causes character set specified in tag to be loaded into terminal memory

CALCULATING

Basic calculating

addl adds 1 to variable specified in tag
subl subtracts 1 from variable specified in tag
zero sets variable specified in tag to zero
define permits author to rename a variable or define a mathematical function for a lesson
calc assigns value of expression on right side of assign arrow to TUTOR variable on left
calcc does one of several possible calculations depending on value of an expression
branch (not a command) when placed in tag field of a continued -calc- permits branch to another line in the continued -calc-

Random numbers

randu provides random numbers, sampled with replacement
setperm sets up two lists of random permutations of integers from 1 through value specified in tag
randp selects randomly from setperm list without replacement
remove removes specified values from one of setperm lists
modperm refills first setperm list with second setperm list

Special calculating

clock puts alphameric string in tag variable giving time on 24 hour clock (not to be confused with the system variable "clock")
date puts character string in tag variable for current month/day/year
day places in tag variable the number of days elapsed since midnight December 31, 1972 to nearest 10^-6 day (approximately 1/10 second).
pack packs specified string, left-justified, in specified variable
compute compiles specified character string, executes, and stores result in specified variable
find searches set of variables for specified character string in a specified position within a variable
search searches set of variables for specified character string of 10 characters, independent of position within a variable
move moves one character from one specified variable and character position to another
movebit moves bit(s) from one specified variable and bit position to another
block copies set of variables into another set
common sets up storage space which is accessible to all students in a particular lesson
unloadc saves temporary variables in lesson common
loadc reloads temporary variables from common to where they are accessible to students

SEQUENCING

Basic sequencing
unit identifies a section of a lesson
entry provides alternate entry point to a unit
base resets main sequence
end terminates branch sequence
arrow places arrow on screen at tag location; waits for student response
long sets maximum length of student response (default is 79 characters)
jkey specifies keys which will initiate judging in addition to -NEXT- key
start indicates following material is to be condensed
stop indicates following material is not to be condensed-- in effect up to next -start- command, if any
* indicates statement on that line is a comment only and is to be ignored by the computer
$$ (not a command) when placed on same line as TUTOR statement indicates that material following is a comment

Author-specified branch
next permits movement to another unit by pressing -NEXT- key (resets base unit if in main sequence)
nextnow terminates processing in a unit and makes only -NEXT- key active
jump causes immediate movement to unit named in tag with change of base to that unit (if in main sequence)
goto causes movement to unit named in tag without change of base but without return to original unit (for exception see full description)
join causes temporary movement to unit named in tag with return to original unit
do causes repeated -join-s of unit named in tag
exit permits termination of -do- or -join- sequence
iferror causes a -goto- to unit in tag if error is found in a -calc-
jumpout jump to specified unit in the lesson named in the tag

Student-initiated branch
help permits branch by pressing -HELP-
helpl permits branch by pressing shifted -HELP-
lab permits branch by pressing -LAB-
labl permits branch by pressing shifted -LAB-
data  permits branch by pressing -DATA-
datal  permits branch by pressing shifted -DATA-
back  specifies unit entered when -BACK- key is pressed
term  permits branching to a unit containing -term- and the string typed as the tag

Timing
pause  delays execution of subsequent statements (minimum .75 second)
time  presses -TIME- key after time specified in tag
press  puts key named in tag into student input buffer
break  interrupts computer if time-slice is about to be exceeded or output-overflow is about to occur.
return  similar to -break- but interrupts processing unconditionally

JUDGING
Always end judging
ok  (no tag) judges ok
no  (no tag) judges no
ignore  (no tag) ignores and erases student response
match  searches student response for occurrence of any of key-words listed in tag

Sometime end judging (m=end if tag matched, e=end if error detected)
answer  checks student response against tag (m)
wrong  checks student response against tag (m)
list  sets up list of synonyms for judging (not executable)
ansv  checks student response against first argument in tag, with tolerance set by second argument (m)
wrongv  same as ansv except for incorrect response (m)
ansva  checks numerical or algebraic student response against first argument in tag, with tolerance set by second argument (m)
vocab  sets up list of ignorable words and synonymous important words (not executable)
concept  checks student response against tag, using vocab list (m)
touch specifies location of touch square (m)
store calculates numerical value of student response and stores it in variable specified in tag (e)
storen searches for and evaluates simple numerical expressions in student responses and stores result in specified variables (e)
exact checks student response with tag for exact character string match (m)
exactc checks student response for exact character string match with one of several arguments in the tag contingent upon the value of an expression

Never end judging

join see description under sequencing
storea stores characters from student response, left-justified, in specified variable (and succeeding variables, if necessary)
specs permits alteration of judging process
or provides alternative answers to a previous judging command
bump removes specified characters from student response before judging
put replaces one character string in student response with another
putd similar to -put- but allows any delimiter between character strings
open places characters of student response, one-by-one, in successive variables starting at tag variable (right-justified)
close takes characters stored in right-most six bits from successive variables and makes judging copy for use by judging commands

Alters judging

judge (not a judging command) alters previous judgment
APPENDIX B

HOW TO PREDICT THE OPERATION OF A TUTOR UNIT

When a TUTOR unit is written, the commands are entered in a list. But when that unit is controlling PLATO, the commands generally do not get used in the order they appear. The chart below can help you predict the sequence of commands during execution of a unit.

<table>
<thead>
<tr>
<th>TUTOR EXECUTION CYCLES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PLATO's Action</strong></td>
</tr>
<tr>
<td>1. Start at -unit-</td>
</tr>
<tr>
<td>Do REGULAR commands.</td>
</tr>
<tr>
<td>If -arrow- is encountered, remember its location.</td>
</tr>
<tr>
<td>Continue processing REGULAR commands.</td>
</tr>
<tr>
<td>Stop at -unit- or JUDGING command.</td>
</tr>
<tr>
<td>1a. If no -arrow- in unit,</td>
</tr>
<tr>
<td>Wait for -NEXT-.</td>
</tr>
<tr>
<td>Start cycle 1 for next unit.</td>
</tr>
<tr>
<td>1b. If -arrow- in unit,</td>
</tr>
<tr>
<td>wait for student response.</td>
</tr>
<tr>
<td>Student responds.</td>
</tr>
<tr>
<td>Student presses -NEXT-.</td>
</tr>
<tr>
<td>2. Start at -arrow-</td>
</tr>
<tr>
<td>Skip REGULAR commands.</td>
</tr>
<tr>
<td>Do JUDGING commands.</td>
</tr>
<tr>
<td>If SPECS in encountered, remember its location.</td>
</tr>
<tr>
<td>Stop at &quot;matched&quot; JUDGING command or -arrow- or -unit-.</td>
</tr>
<tr>
<td>PLATO's Actions</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3. Start immediately at &quot;matched&quot;</td>
</tr>
<tr>
<td>JUDGING command.</td>
</tr>
<tr>
<td>Do REGULAR commands.</td>
</tr>
<tr>
<td>Stop at JUDGING command or -arrow- or -unit-.</td>
</tr>
<tr>
<td>4. Start immediately at -specs-.</td>
</tr>
<tr>
<td>Do REGULAR commands.</td>
</tr>
<tr>
<td>Stop at JUDGING command.</td>
</tr>
<tr>
<td>4a. If judgement was &quot;no&quot;, wait for student to change response.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Start cycle 2.</td>
</tr>
<tr>
<td>4b. If judgement was &quot;OK&quot;.</td>
</tr>
<tr>
<td>Start cycle 5.</td>
</tr>
<tr>
<td>5. Start at -arrow-.</td>
</tr>
<tr>
<td>Seek -arrow- or -unit-.</td>
</tr>
<tr>
<td>5a. If -arrow-, Remember location of -arrow-; Do REGULAR commands.</td>
</tr>
<tr>
<td>Stop at JUDGING command.</td>
</tr>
<tr>
<td>Wait for student response.</td>
</tr>
<tr>
<td>Start cycle 2.</td>
</tr>
<tr>
<td>5b. If -unit-, Wait for -NEXT-.</td>
</tr>
<tr>
<td>Start cycle 1 for next unit.</td>
</tr>
</tbody>
</table>
APPENDIX C

Aids for Authors

There are several lessons intended to assist authors. They are available from any terminal. Some of them are listed below:

<table>
<thead>
<tr>
<th>Lesson Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) graphd</td>
<td>How to write and draw on the screen. Includes writing in various sizes, drawing lines and circles, etc.</td>
</tr>
<tr>
<td>(b) calcd</td>
<td>How to use TUTOR variables to keep records and perform calculations.</td>
</tr>
<tr>
<td>(c) tutor</td>
<td>Description of the most commonly used TUTOR commands.</td>
</tr>
<tr>
<td>(d) aid1</td>
<td>General reference on PLATO and TUTOR.</td>
</tr>
<tr>
<td>(e) aid2; aid3</td>
<td>Reference manuals for the TUTOR language.</td>
</tr>
<tr>
<td>(f) grafpack</td>
<td>Reference manual for special graph-making commands.</td>
</tr>
<tr>
<td>(g) service</td>
<td>Helps authors generate complicated displays and individual characters.</td>
</tr>
<tr>
<td>(h) charset</td>
<td>Helps authors create new character sets.</td>
</tr>
</tbody>
</table>

Help is also available from the staff of CERL; do not hesitate to ask for assistance.

Announcements about changes in TUTOR and answers to users' questions appear in the lesson space called notes. (Enter notes via the editor, not as a student.) You may enter questions in notes. They will usually be answered within 24 hours. (The change code is notes.)
APPENDIX D

Using the TUTOR Editor

The editor for TUTOR lessons is itself a TUTOR lesson, and has many aids for TUTOR authors incorporated into its structure. At critical points, if you become momentarily confused about what you may do next, you may usually obtain aid by pushing the -HELP- key. Consequently there is little need for a comprehensive printed guide to TUTOR editing. This brief note is intended to guide new TUTOR authors who have not learned yet what key to push to get from "here" to "there".

SIGNING ON AS AN AUTHOR

1. When you sit down at the terminal, if the screen is blank, or if you see anything except the Press NEXT to Begin message, press the -BACK- key until this message appears. (If the -BACK- key does not bring you the Press NEXT message, press the shifted -STOP- key then press the -BACK- key several times.)

2. Press -NEXT- to go to the SIGN-ON page.

3. On the SIGN-ON page, type your name and press -NEXT-. Then in the space saying "Type Course Name" type instead the word "author" (or just "a") and press -NEXT-. Type your own ID code (whatever you wish), and push -NEXT- to go to EDIT OPTION page.

TO REQUEST A NEW LESSON

1. On the EDIT OPTION page, type "request" and press -NEXT-. Follow directions on the REQUEST page to get to the NEW LESSON page.

2. On the NEW LESSON page, enter your name, the name of the lesson you wish to create (up to 10 characters), and your telephone number.
3. Later that day, the machine operator will create the lesson for you if you have permission.
4. You may check to see if your lesson has been created after a few hours, by going to the EDIT OPTION page as specified in the SIGN-ON procedure, then typing the desired lesson name. If you receive a NO SUCH LESSON message, the lesson has not yet been created. Have patience.

TO SPECIFY DESCRIPTIVE INFORMATION ABOUT YOUR LESSON

1. The first time you enter your lesson by typing its name on the EDIT OPTION page, you should fill in the descriptive data requested on the DATA page. Type the number corresponding to the datum you wish to enter, type the information requested, then push NEXT. When you are through with this DATA page, push BACK to begin editing your lesson on the LESSON EDIT page. Type your last name first, (e.g., Bitzer, Donald L.), so that lessons can automatically be alphabetized by authors' names. Data like your name, department, and phone number are obvious. The "change code" is the security code someone must use to edit or change the lesson. Make sure you remember it or you will not be able to edit your own lesson without help from the machine operator!! If you leave the "change code" blank, anyone may edit your lesson. The "inspect code" is the security ID code anyone must have to look at your lesson. If you don't mind who sees it, leave it blank. Finish with a brief one-line description of your lesson. (You may change any of this data later by entering your lesson and pushing DATA.)

2. When you reach the EDIT OPTION page, type the name of your lesson and push NEXT. You will reach the LESSON EDIT page which lists the name of your lesson, the time, and letters a-n specifying one or more "blocks" of lesson text contained in your lesson.
When you enter a brand-new lesson (after specifying the descriptive data) there will be only one block (labeled "a") with approximately 300 words (computer words of 10 character codes each) of space in it.

Lessons are divided into blocks to optimize edit time and memory space requirements. Each block may contain up to about 60 lines of 50 characters each, or (more typically) about 100 lines of 30 characters each.

You may give the blocks descriptive names so that you can more easily remember what is in which block. Several blocks may have the same name if you wish.

TO EDIT A SPECIFIC BLOCK: Push the letter corresponding to the block label, and you will be taken to the BLOCK EDIT page.

TO START A NEW BLOCK: If you wish to start a new block after block "a", push SHIFT-A. You will be sent to the NEW BLOCK NAME page where you may enter a name for that block, then you press NEXT to go to the BLOCK EDIT page.

TO DESTROY A BLOCK: There is no special option for this. The block will automatically disappear when all text has been deleted from it.

TO COPY A BLOCK: Use the SAVE option on the BLOCK EDIT page, start a new block, and INSERT the SAVE'd lines into the new block.

Lessons with many blocks may be split into PARTS because there are too many blocks to show on one LESSON page. To go to the PART 2 page from PART 1, push the + key. Push the - key to return to PART 1 from the PART 2 page.

EDITING A BLOCK OF LESSON TEXT

The lesson and block name will appear at top left and center of the BLOCK EDIT page, along with the space (in computer words) left in that block. You may enter editing commands from the keyboard which will appear on the screen at the top left.
2. EDIT COMMANDS

(Enter command, then push NEXT)

- 4D -

f4 roll the display forward (up) 4 lines
b20 roll the display back (down) 20 lines
i3 insert new lesson text after line 3 on the screen
   (Important: use TAB key to separate command and tag.)
r5 replace lesson text starting with line 5 on the screen
d50 delete 50 lines starting from the one at the top
   of the screen
   (to activate this option, you must push SHIFT-HELP)
s14 save 14 lines starting from top of screen and put
   in "save buffer".
is2 insert the "saved" lines after line 2 now on the
   screen.
u proceed to the next unit and display it starting
   at line 1.
uwhee proceed to unit "whee" (if it is in this block)
pnext proceed to first following line containing a
   TUTOR "next" command. Any characters may follow
   the p (e.g., Pca - proceed to next line which
   starts with "ca")
xdog search block for the next line containing "dog"
   and bring it to top of screen. (any character
   string may follow the x) The search is "circular"
   through the block.
+ or - roll screen up or down one line
x of divide roll screen up or down ten lines
space continue filling screen with more lines of text
   following the line at which it stopped.
$7 Show 7 lines of text on the screen before stopping.
   (Normal option is 10 on entering lesson).
out escape from a major blunder (like deleting too
   much) without transferring edits you have made in
   this block back to disk.

3. Pressing NEXT without entering any edit command will repeat
   most previous commands. If no number is entered after i,f,b,
   d, or r, a one is understood. Therefore a "f" command by
   itself meanf "fl".
USING THE COPY AND EDIT KEYS

1. The -COPY- and -EDIT- keys work as they do in TUTOR lessons.
2. The -COPY- key (having a carat on the keycap A) will copy into a line being "inserted" or "replaced" from the line appearing immediately above it on the screen. Each push of the -COPY- key will copy another "word" (any characters bounded by spaces or punctuation) into the line being worked on. At any point, pushing the SHIFT-COPY key will copy the rest of the line.
3. The first push of the -EDIT- key (having a square on the keycap) will wipe out the entire line being worked on. Successive pushes of -EDIT- will return "words" to that line one at a time, in a fashion similar to the COPY key. SHIFT-EDIT copies in the remaining words in the line.
4. The -ERASE- key erases one character at a time. The SHIFT-ERASE key erases one word in the worked-on line at a time.

TESTING THE LESSON YOU ARE EDITING

1. Pressing SHIFT-STOP while editing a lesson will condense that lesson and send you into the lesson as a student. When finished testing your lesson, press SHIFT-STOP to again return to editing without passing through the entire sign-on sequence again.
2. If you sign on as a student (type "student" or "s" on the SIGN-ON page), you may test or demonstrate any lesson by typing its name on the TUTOR display page.
3. To enter the editor from the TUTOR page, type "edit" instead of a lesson name.
4. Press -p- on the LESSON EDIT page to specify which blocks are to be condensed and which are not.
GENERAL INFORMATION

1. Pressing -BACK- will take you from the BLOCK EDIT to LESSON EDIT to EDIT OPTION page to ENTER ID page to TUTOR page to Press NEXT to Begin. Back out all the way when finished to prevent anyone from using your ID and editing YOUR lesson inadvertently.

2. Pressing -STOP- while text is being displayed will immediately stop the display and await a new command.

3. To request a printout, type "request" on the EDIT OPTION page and follow instructions.

4. If you have comments about the behavior of TUTOR or general messages of interest to other authors, type "notes" on the EDIT OPTION page and enter your note in the latest block, just as if you were editing a lesson.