DOCUMENT RESUME

ED 124 137 IR 003 525

AUTHOR	Ghesquiere, James R.; And Others
TITLE	Introduction to TUTOR.
INSTITUTION	Illinois Univ., Urbana. Computer-Based Education Lab.
SPONS AGENCY	Advanced Research Projects Agency (DOD), Washington, D.C.; National Science Foundation, Washington, D.C.
PUB DATE	Jul 75
CONTRACT	US-Army/DAHC-15-73-C-0077; USNSF-C-723
NOTE	163p.
AVAILABLE FROM	PLATO Publications, Computer-based Education Research Lab, 252 Engineering Research Laboratory, University of Illinois, Urbana, Illinois 61801 ($2.70, prepayment required)
EDRS PRICE	MF-$0.83 HC-$8.69 Plus Postage.
DESCRIPTORS	*Computer Assisted Instruction; Higher Education; Instructional Systems; *Manuals; *Programing Languages
IDENTIFIERS	*PLATO IV; Programmed Logic for Automated Teaching Operations; TUTOR

ABSTRACT

This manual provides instruction in the use of TUTOR, the computer language used in the PLATO computer based education system. A preliminary lesson and the first seven chapters introduce the basic features of the language, and later chapters provide a more comprehensive understanding of the concepts involved. The text explains how to use the language to direct the computer to present displays, pose questions, judge responses, perform calculations, determine interconnections between units, and to branch to units relevant to user responses. The appendixes provide a summary of TUTOR commands and show procedures for editing in the TUTOR language. (EMH)

Introduction to TUTOR

James R. Ghesquiere

Celia R. Davis

Charlene A. Thompson

PLATO Services Organization

Computer-based Education Research Laboratory
Urbana, Illinois

First Printing, March 1974
Second Printing, June 1974
Third Printing, July 1975

This manuscript was prepared with partial support from the National Science Foundation (USNSF C-723) and the Advanced Research Projects Agency (US Army/DAHC 15-73-C-0077).

ACKNOWLEDGMENT

The authors wish to thank Elaine Avner, R.A. Avner, Robert Bohn, Larry Francis, William Golden, Bruce Sherwood, and Judy Sherwood for reading the manuscript and making many valuable suggestions. The authors would like to express their appreciation to Sibyl Pellum and Terri Parker for typing the manuscript, and to Wayne Wilson for doing the graphic work. We are also grateful to those PLATO authors who used and commented upon early drafts of this manual.

Table of Contents

PREFACE

TUTOR is the name of the computer language used by the PLATO computer-based education system. This book is an introduction to the TUTOR language for persons who have no knowledge of computers. After working through the training program of which this book is a part, you will be able to write useful and successful lessons and will be prepared to learn aspects of TUTOR not formally introduced here.

This training program has three parts: this book, a PLATO lesson named "introtutor", and programming exercises for you to do in your lesson space. Lesson "introtutor" and chapters 1 through 7 introduce the basic features of TUTOR. Chapters 8 through 12 give an expanded and more comprehensive view of the material presented in the earlier chapters. Chapter 13 contains information on how to proceed after completing this book. On the back cover is a checklist to guide you as you work through the book, lesson "introtutor", and the exercises.

1. Writing on the Panel and Judging Student Responses

The TUTOR Building Block, the Unit

TUTOR lessons are composed of units. A unit consists of statements directing the computer to do any or all of the following:

present a display (text, graphics, slides) on the panel
pose a question or problem
judge various responses as "ok" or "no"
perform calculations
determine how the present unit connects to other units in
    the lesson

All statements in TUTOR have the form

```
command tag
```

where the command is a general instruction to the computer and the tag is specific information for performing the general instruction.

Each unit is initiated by a -unit- command. (Hyphens placed around words in this manual are used for clarity in identifying TUTOR commands. The hyphens are not part of the command.) The tag of a -unit- command is the name of the unit. You may give a unit any name you wish but it may not be longer than 8 characters, and no two units in a lesson may have the same name. The end of a unit is indicated by PLATO's encountering another -unit- command.

```
unit    writing      ─┐  unit
at      1415          │  "writing"
write   How are you? ─┘
unit    question     ─┐  unit
.                     │  "question"
.                     │
.                    ─┘
```

The first command, -unit-, marks the beginning of the unit. The tag "writing" is the name of the unit. The statement -unit question- marks the end of unit "writing" and the beginning of unit "question". The -write- command instructs the computer to put writing on the panel. Thus the words "How are you?" will be displayed on the panel. The -at- command, preceding the -write- command, causes the tag of the following -write- command to be displayed at a particular location on the panel. The tag "1415" of the -at- command specifies the location of the writing.

The panel has 32 horizontal lines and each line has 64 spaces. Any location on the panel is specified by indicating the number of lines from the top and the number of spaces from the left. Thus the statement

```
        at      1415
```

designates the fourteenth line from the top, fifteen spaces from the left. The text in figure 1.1 is written beginning at location 1415. This location system is called the "coarse grid". A way of locating points more precisely, the "fine grid", will be discussed later.

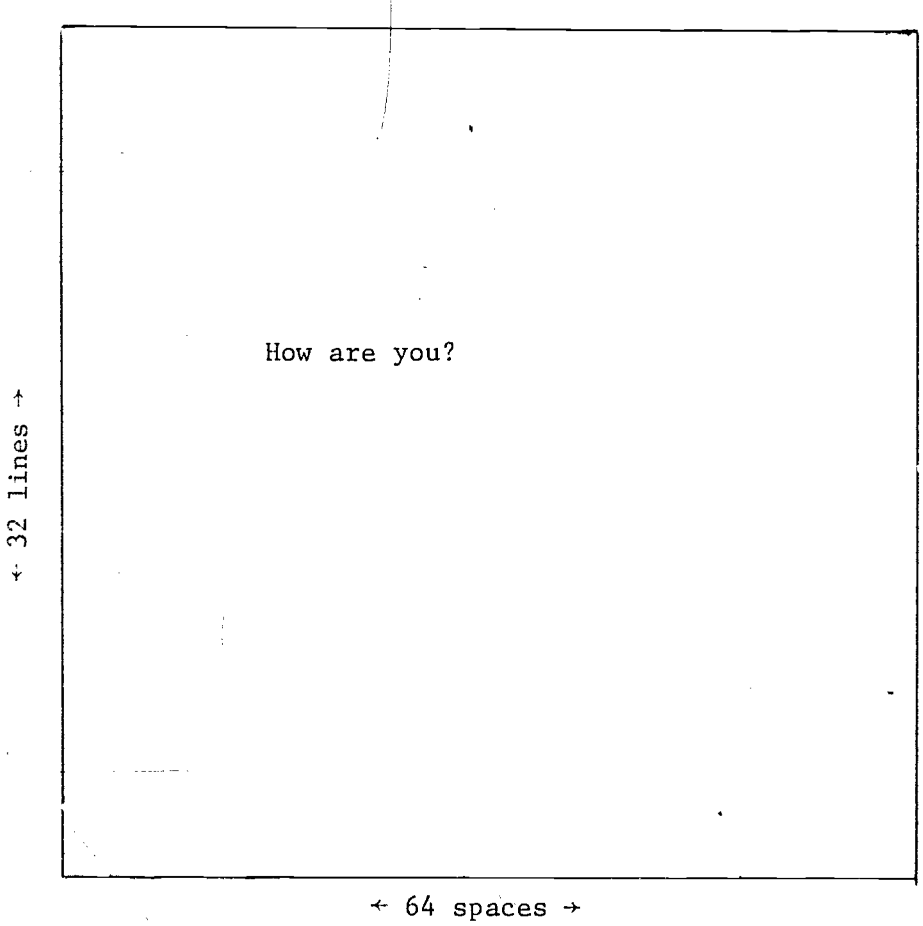


Figure 1.1 The text appears at line 14, space 15

Suppose you want to write "How are you?" on the fourteenth line at the fifth space. Use the statement

```
          at        14Ø5
```

PLATO always interprets the last two digits as the space reference. If there are three digits, the first one is interpreted as the line reference; if there are four digits, the first two are interpreted as the line reference. The statement

```
          at        145
```

causes the writing to begin at the 45th space of the first line.

Accepting Student Responses

In any TUTOR lesson you will probably want to ask questions of the student, and to tell him whether his response is right or wrong. Thus you need a way to accept student input and to evaluate it. The following unit presents a question and specifies a correct answer.

```
          unit      gorge
          at        12Ø3
          write     Where is Louis S. B. Leaky's anthropological dig?
          arrow     14Ø1
          answer    Olduvai Gorge
```

The -arrow- command signifies that a student response is expected. An arrow is plotted at the location specified in the tag (in this case 14Ø1). The format of the -arrow- tag is the same as the -at- tag. In this unit the arrow is plotted 14 lines from the top and 1 space from the left. When the student types a response it will appear on the panel to the right of the arrow.

After the student types a response and presses the NEXT key, his response is compared to the tag of the -answer- command. (Names of keys on the PLATO keyset appear in full capital letters in this manual.) If they are the same, the word "ok" is written on the panel to the right of the response. If they are not the same, several things happen automatically. If the student misspells a word, it is underlined (_______). If he has a word out of order, the marking (←) appears. If he has words in his response which don't occur at all in the -answer- tag, x's appear under those words ($_{\text{xxxxxxx}}$). The word "no" appears to the right of the response and PLATO waits for the student to press NEXT or ERASE and try again.

Spaces and punctuation marks in the student's response are all interpreted as word separators. If the student should leave two spaces between the words "Olduvai" and "Gorge", his response would still be judged "ok". However, for the student to be judged "ok" he must type "Olduvai Gorge" with both words capitalized. If he types "Olduvai gorge" it will be judged "no" because he didn't capitalize the word "gorge". Unit "gorge" can be modified to accept either small or capital letters. The -specs- command allows you to specify judging options. If we insert the statement

```
specs   bumpshift
```

before the -answer- command, and change the -answer- tag to

```
answer  olduvai gorge
```

(that is, no capitalization of "olduvai" or "gorge" in the -answer- tag) then the student may type either small or capital letters and his response will be judged "ok". The tag "bumpshift" means that capitals will be ignored in the student response. Several other -specs- tags are available, such as

```
specs   okspell
```

which judges "ok" even if words are misspelled in the student's response. Another -specs- tag, "nookno", will inhibit the writing of the "no" or "ok" after the student's response.

Even with these modifications to unit "gorge", many correct answers will still be judged "no". For instance, "It is Olduvai Gorge in Kenya." or "The Olduvai Gorge." would be judged "no". To allow for this, we can make further modifications to the -answer- tag:

```
answer  <the,it,is,in,at,kenya> olduvai gorge
```

The words within < > are "ignorable" words. If any of them appear anywhere in the student's response, they will be ignored by PLATO; they will not cause the answer to be judged "no". The words "olduvai" and "gorge" are called "required" words. They must appear in the student response and in the same order as they appear in the -answer- tag for the response to be judged "ok".

We might want to make one more alteration of the -answer- tag. If "Olduvai canyon" is an acceptable response, as well as "olduvai gorge", we can specify that "canyon" and "gorge" are synonyms for purposes of this answer.

```
answer <the,it,is,in,at,kenya> olduvai (gorge,canyon)
```

Words enclosed in ( ) are called "synonomous words" or just "synonyms". Now the required words for this answer are "olduvai" and <u>either</u> "gorge" or "canyon".

We have already seen that when a student response does not match the -answer- tag, some feedback may appear (===,xxx, ←). But sometimes you may want to give a student more specific feedback if he makes a particular error.

More additions to unit "gorge" are shown below. The tag for the last -write- statement in the modified unit "gorge" is displayed as it looks in the TUTOR editor, which has shortened lines for display (since the command must appear with each statement). That is, the "ees." wraps around to the next line. When displayed for a student, the phrase "That's the ... chimpanzees." will appear on one line. In general, all examples of TUTOR code which are provided in this manual appear as if they were written in the TUTOR editor.

```
unit    gorge
at      12Ø3
write   Where is Louis S. B. Leaky's anthropological dig?
arrow   14Ø1
specs   bumpshift,okspell
answer  <the,it,is,in,at,kenya> olduvai (gorge,canyon)
wrong   <it,is,the,at,in,tanzania> gombe stream research cen
        ter
write   That's the site of Jane Goodall's work with chimpanz
        ees.
```

If the student types "It is at the Gombe Stream Research Center in Tanzania." or "the Gombe Stream Research Center" his response will be judged "no" and the sentence "That's the site of Jane Goodall's work with chimpanzees." will appear automatically starting 3 lines below the arrow.

The -wrong- command works like -answer- except that if it is matched the response is judged "no". The words in < > are "ignorable" words; the rest are required words. The student must have the words "gombe stream research center" in that order for his response to match the -wrong-. He may have any or all of the "ignorable" words in any order, and his response will still match the -wrong-. The sentence "That's the site of Jane Goodall's work with chimpanzees." will appear only if the student response matches the -wrong- tag. Any -write- commands following a -wrong- or an -answer- statement are done only if the student's response matches that -wrong- or -answer- tag. If the student's response matches neither the -answer- tag nor the -wrong- tag, the response will be judged "no". PLATO automatically judges "no" any response not anticipated by the author. Figure 1.2 illustrates this process pictorially in a flow chart of unit "gorge".

If the student's response matches the -answer- tag, the rest of the unit is not done, because all the rest of the commands specify how to handle incorrect responses. Since the response is correct, there is no need to do that. Only if his response matches the -wrong- tag will PLATO execute the command following the -wrong- (i.e. write "That's the site of Jane Goodall's work with chimpanzees."). Then PLATO judges the response "no" and waits for another student response. If the response matches neither the -answer- nor the -wrong- tag, PLATO writes its default "no" and goes back and waits for another response. This will happen as many times as the student types an incorrect answer; only when the arrow is satisfied by a correct student response will PLATO allow the student to continue.

Added to unit "gorge" below is a -write- statement following the -answer-.

```
unit    gorge
at      12Ø3
write   Where is Louis S. B. Leaky's anthropological dig?
arrow   14Ø1
specs   bumpshift,okspell
answer  <the,it,is,in,at,kenya> olduvai (gorge,canyon)
write   Homo habilis was discovered there.
wrong   <it,is,the,at,in,tanzania> gombe stream research cen
        ter
write   That's the site of Jane Goodall's work with chimpanz
        ees.
```

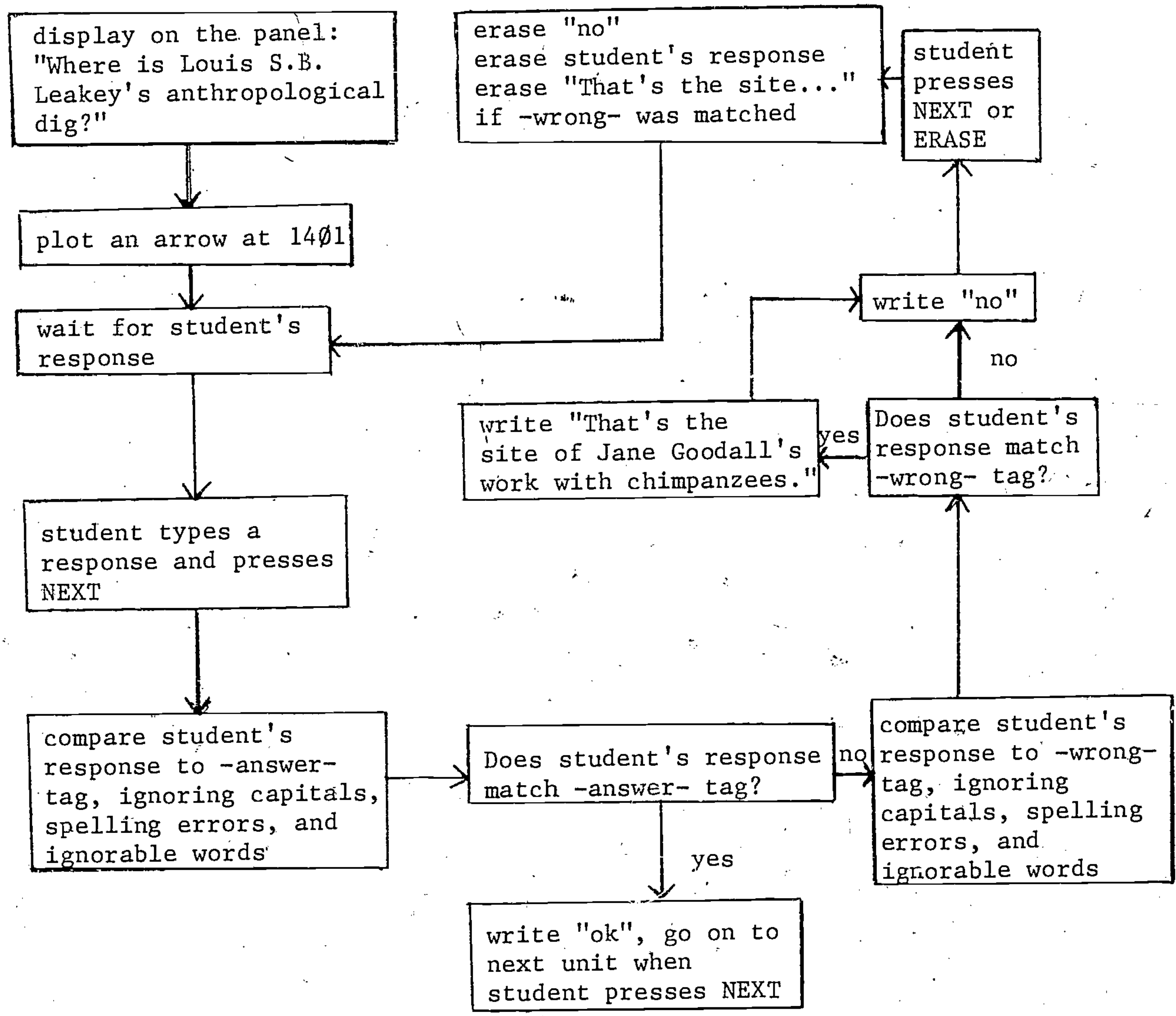
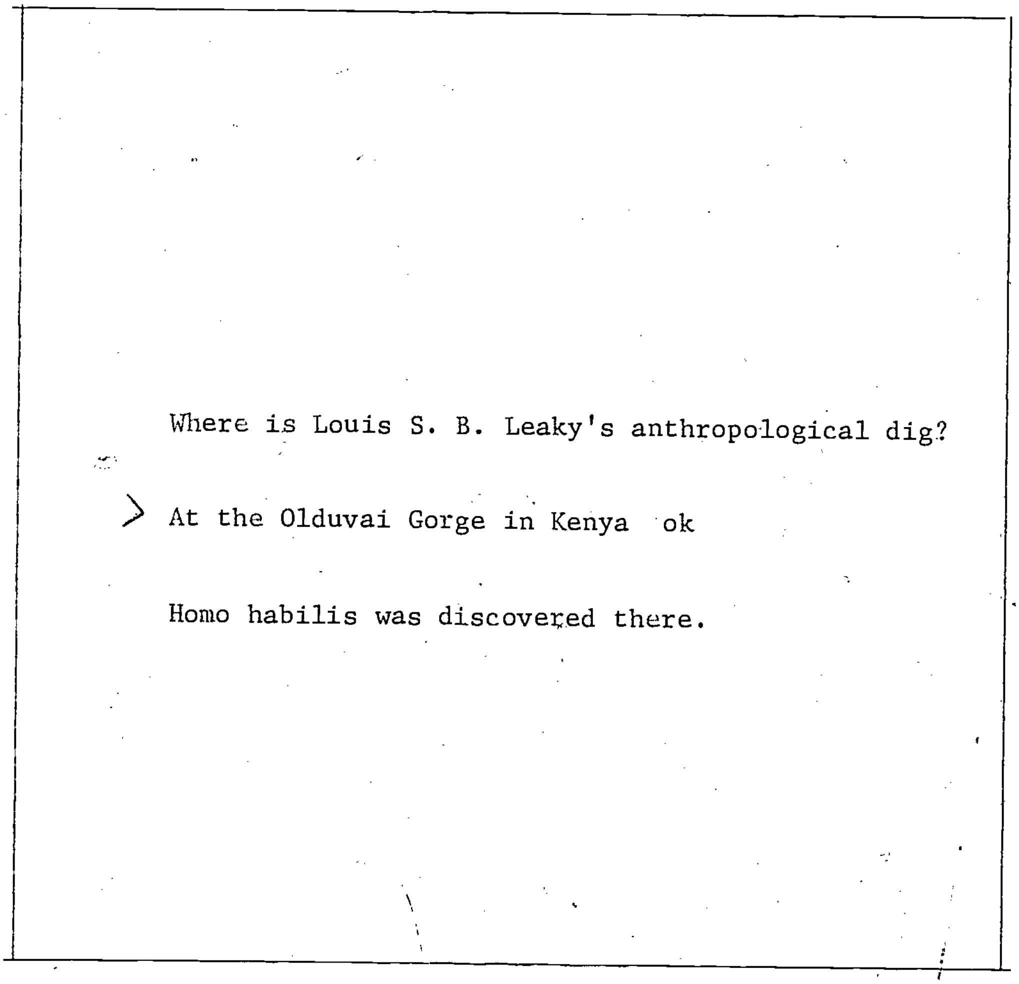

display on the panel: "Where is Louis S.B. Leakey's anthropological dig?"
plot an arrow at 14Ø1
wait for student's response
student types a response and presses NEXT
compare student's response to -answer- tag, ignoring capitals, spelling errors, and ignorable words
Does student's response match -answer- tag?
yes
write "ok", go on to next unit when student presses NEXT
no
compare student's response to -wrong- tag, ignoring capitals, spelling errors, and ignorable words
Does student's response match -wrong- tag?
yes
write "That's the site of Jane Goodall's work with chimpanzees."
no
write "no"
student presses NEXT or ERASE
erase "no"
erase student's response
erase "That's the site..." if -wrong- was matched

Figure 1.2 Flow diagram of unit "gorge"

The sentence "Homo habilis was discovered there." will appear after the student has correctly answered the question. Figure 1.3 shows how this unit looks on the student's panel after a correct response.

There is no -at- command in unit "gorge" between the -answer- and -write- or between the -wrong- and -write- statements. Such writing will automatically be placed 3 lines below the response. If you want to position it elsewhere, insert an -at- command with the desired tag before the -write-.



```
Where is Louis S. B. Leaky's anthropological dig?

> At the Olduvai Gorge in Kenya  ok

Homo habilis was discovered there.
```

Figure 1.3 "ok" response to unit "gorge"

The -answer- and -wrong- tags of unit "gorge" have several ignorable words. Even with this list, not all correct responses are accepted as "ok" by PLATO. For example, the response "It is located in Olduvai Gorge" would be judged "no" because the word "located" is not listed among the anticipated words. There is a -specs- tag, "okextra", which permits extra words in the student response. When a -specs okextra- statement is used, only the required words of the -answer- and -wrong- commands need to be specified. Unit "gorge" can be modified by adding the "okextra" tag to the -specs- statement and removing the ignorable words from the tags of the -answer- and -wrong-.

```
unit    gorge
at      12Ø3
write   Where is Louis S. B. Leaky's anthropological dig?
arrow   14Ø1
specs   bumpshift,okspell,okextra
answer  olduvai (gorge,canyon)
write   Homo habilis was discovered there.
wrong   gombe stream research center
write   That's the site of Jane Goodall's work with chimpanz
        ees.
```

Because capitalization and extra words are often not important in a student's response, many -arrow- commands will have a -specs bumpshift,okextra- associated with them.

Exercise A - General Comments

When doing these exercises you will be editing in whatever lesson has been assigned to you. The name of your lesson was decided by you and your course director. Whenever you are asked to do some editing in your lesson, you should enter on the author mode display the lesson name you and your course director picked.

To help you in these exercises there are sample lessons available through lesson "introtutor" which show you what your lesson will resemble in student mode. You cannot edit the sample lessons; you can only use them as a student. Enter lesson "introtutor" again in student mode and choose sample lesson A. Study sample lesson A noticing the flow of units and the interaction between the computer and the student.

You see (in sample lesson A) that exercise A consists of four TUTOR units. A flow diagram of units you will create in this exercise is presented in figure 1.4. In these flow diagrams, each unit is represented by a box. The name of the unit is in the box and if the unit requires a response from the student, an arrow (>) is in the box. The lines connecting units indicate which unit the student will see after completing the present unit and which key press is required to branch to another unit (NEXT). For easy cross reference between your own lesson and exercise descriptions, use the unit and block names suggested in the exercises.

The topics for the exercises are science and psychology. Two different pedagogical approaches have been used: simulation in the science topic and tutorial in the psychology topic. Both topics are summarized below in order to provide some background material.

Newton: Isaac Newton (1642-1727) can be considered the father of classical physics. He developed three laws of motion, that is, mathematical equations which describe how objects' movement through space is related to the forces that move them. The most often quoted story about Newton is that one day, while he was sitting under an apple tree, an apple fell and struck him. The event supposedly "brought the light" to Newton about some ideas on motion and gravity.

Transactional Analysis: The specific material for this topic is Thomas Harris' book I'm OK -- You're OK. Harris and Eric Berne, the author of Games People Play, are the popularizers of transactional analysis. They say that their psychology is different from Freud's because it is usable by non-psychologists. The questions in the psychology topic analyze which components of people (Parent, Child, or Adult) are active in transactions.

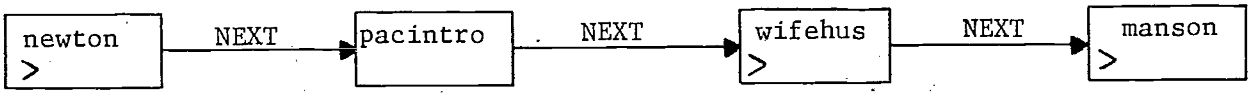


Figure 1.4 Flow diagram of units for exercise A

Physics Units

Enter your lesson from the author mode display and delete all TUTOR code which might be in it. When only block "a" remains and all lines have been deleted from it, your lesson is empty. Now you are ready to start inserting your own material. You can refresh your memory on how to edit by pressing HELP while editing or by referring to Appendix B.

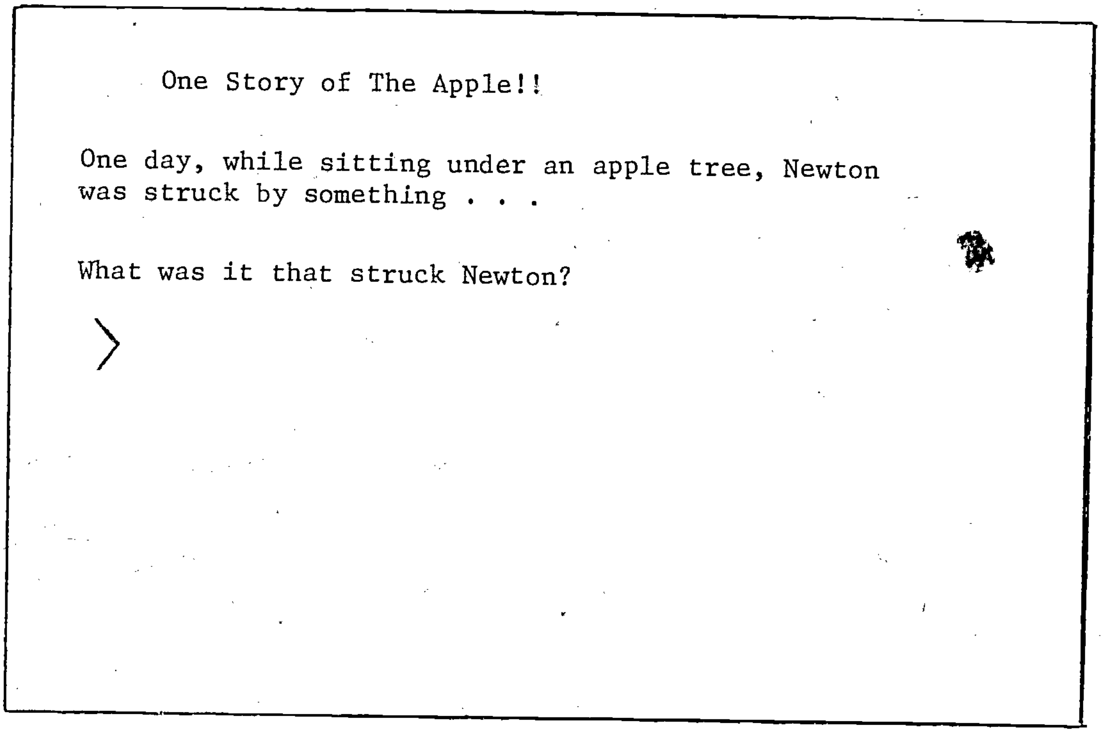
Create a new block after "a" (shift "a" while on the block listing display enables you to add a block after "a"). Call the new block "physics". Insert the code from figure 1.5 in block "physics". Remember to use the TAB key to line up the tag field correctly. You should not type the line numbers. These numbers appear automatically after you have entered TUTOR code and have returned to the line display. After this code from figure 1.5 is entered, press SHIFT-STOP (both keys together) to condense your lesson. Your display panel should resemble figure 1.6.

Try typing some responses not anticipated by your -answer- command, such as "apple", "a thought", "He was struck by an idea". This version of your lesson accepts as correct only the responses "idea" and "an idea". All other responses are judged "no".

Edit your lesson again and expand the response judging capabilities of unit "newton" so that it accepts the above responses. Modify the -answer- command to include synonyms for "idea" and add the appropriate ignorable words. When a -specs bumpshift- is used, there should be no capital letters in the tags of -answer- and -wrong- commands, so check that you have none.

```
unit    newton
at      21Ø
write   One Story of The Apple!!
at      4Ø5
write   One day, while sitting under an apple tree, Newton
        was struck by something . . .

        What was it that struck Newton?
arrow   1Ø1Ø
answer  <an> idea
at      12Ø5
write   Yes, it was while under an apple tree (and maybe eve
        n
        being struck by a falling apple` that Newton cleared
        up
        some ideas on gravity.
```

Figure 1.5 Code for unit "newton"



```
     One Story of The Apple!!

One day, while sitting under an apple tree, Newton
was struck by something . . .

What was it that struck Newton?

>
```

Figure 1.6 Display from unit "newton" seen in student mode

Try your lesson again in student mode. The previously unaccepted responses are now judged "ok". Complete unit "newton" so that it resembles figure 1.7.

Condense your lesson and try the new responses. Enter the wrong response first and notice that only the last comment written by PLATO is erased when you press NEXT or ERASE to enter another response. The other comment remains on the panel. After a "no" judgment by PLATO, only the last -write- statement is erased.

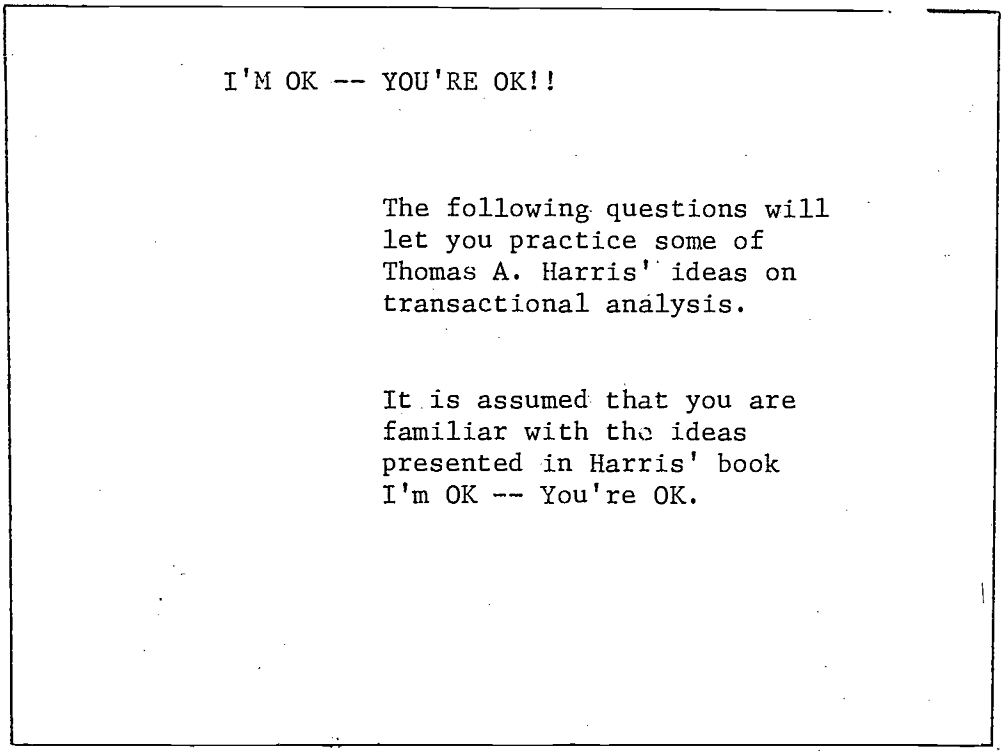
```
unit    newton
at      21Ø
write   One Story of The Apple!!
at      4Ø5
write   One day, while sitting under an apple tree, Newton
        was struck by something . . .

        What was it that struck Newton?
arrow   1Ø1Ø
specs   bumpshift,okextra
answer  apple
at      12Ø5
write   That is the legend.  From this accident, Newton supp
        osedly
        crystallized some ideas on gravity.
answer  (idea,thought)
at      12Ø5
write   Yes, it was while under an apple tree (and maybe eve
        n
        being struck by a falling apple) that Newton cleared
        up
        some ideas on gravity.
wrong   eve
at      2Ø2Ø
write   Try again, for what hit
        Newton under the APPLE
        tree (apples fall . . .)
at      12Ø5
write   NO!!  You are in the wrong era!
```

Figure 1.7 Code for unit "newton" with more response judging

Psychology Units

While in sample lesson A, try responses of "parent", "adult" and "child" at the arrows in the psychology units. By trying all responses at each arrow you might be able to anticipate what commands will be needed in your lesson to achieve the same results.

Specific directions for the TUTOR code to be inserted in your lesson will not be given for the psychology units. Enter your lesson as an author. Create a new block called "psychology" after block "physics" (shift "b" to do this). In block "psychology" create a unit called "pacintro" which looks like figure 1.10.



I'M OK -- YOU'RE OK!!

The following questions will
let you practice some of
Thomas A. Harris' ideas on
transactional analysis.

It is assumed that you are
familiar with the ideas
presented in Harris' book
I'm OK -- You're OK.

Figure 1.10 Display for unit "pacintro"

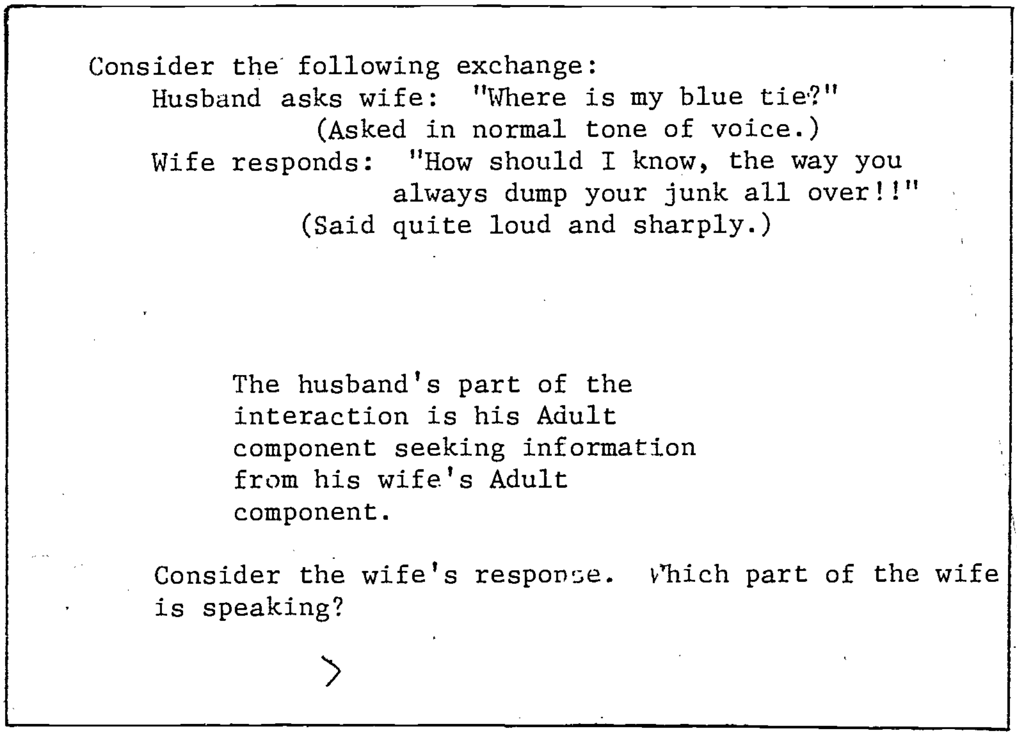
Create another new block after block "psychology" called "wifehus" (for WIFE and HUSband). In this block insert the necessary code to reproduce the wife-husband transaction you saw in sample lesson A. Call the unit in your new block "wifehus". Figure 1.11 shows the display you want to produce and figure 1.12 summarizes the response judging. A structure similar to unit "newton" is needed for the psychology units with arrows.

The last psychology unit (man-son transaction) should be inserted in another new block called "manson". In this new block, which should follow block "wifehus", insert a unit named "manson" which has the commands to produce the last psychology unit. Figures 1.13 and 1.14 show the display and response judging for unit "manson".

To complete this exercise, do all the "checks" described in figure 1.15. These checks constitute a method for catching subtle errors in your lesson. There are three areas you should consider:

layout of all displays
response judging
sequence of units

Checking your response judging requires trying not only anticipated correct and incorrect responses, but also some responses which are obviously incorrect.

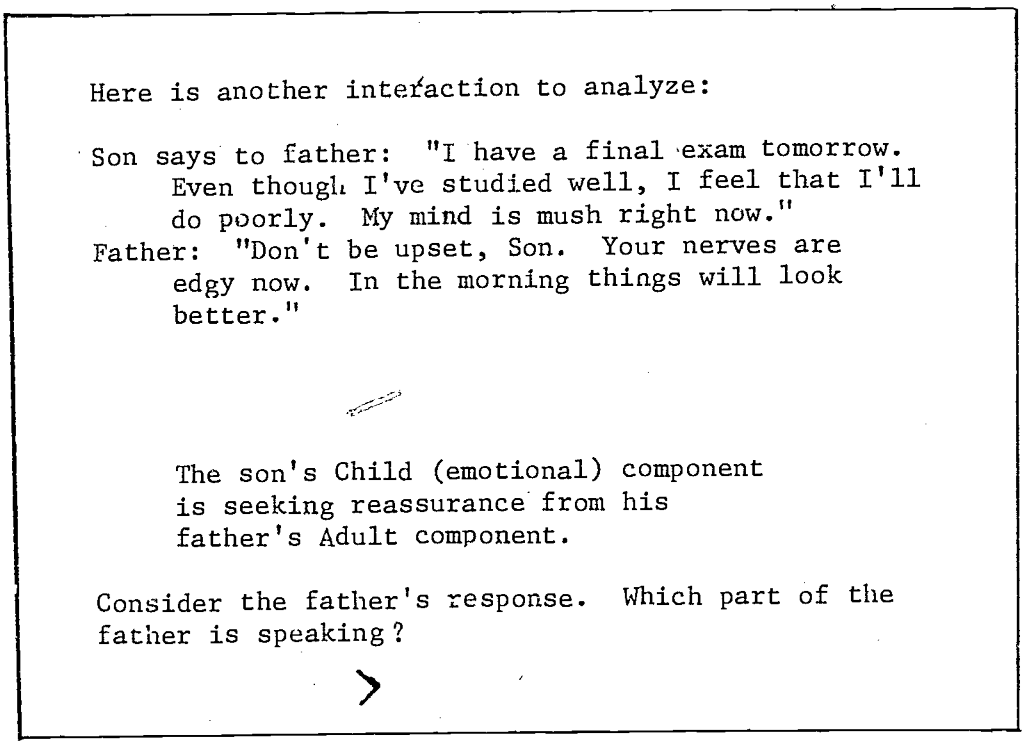


```
Consider the following exchange:
    Husband asks wife:  "Where is my blue tie?"
              (Asked in normal tone of voice.)
    Wife responds:  "How should I know, the way you
                     always dump your junk all over!!"
             (Said quite loud and sharply.)



          The husband's part of the
          interaction is his Adult
          component seeking information
          from his wife's Adult
          component.

    Consider the wife's response.  Which part of the wife
    is speaking?

              >
```

Figure 1.11 Display produced by unit "wifehus"

Student Response	PLATO's Message	PLATO's Judgment
parent	Exactly right.	ok
adult	Her response (sarcastic tone) contains more than information about the tie.	no
child	If it were the Child, it would be more self-centered.	no

Figure 1.12 Summary of response judging for unit "wifehus"



Here is another interaction to analyze:

Son says to father: "I have a final exam tomorrow.
Even though I've studied well, I feel that I'll
do poorly. My mind is mush right now."
Father: "Don't be upset, Son. Your nerves are
edgy now. In the morning things will look
better."

The son's Child (emotional) component
is seeking reassurance from his
father's Adult component.

Consider the father's response. Which part of the
father is speaking?

>

Figure 1.13 Display from unit "manson"

Student Response	PLATO's Message	PLATO's Judgment
adult	Good. His response is neither emotional (Child) nor judgmental (Parent).	ok
child	There is no emotional aspect to the response	no
parent	A Parent's response would have scolded the son for being afraid or not trying hard enough.	no

Figure 1.14 Summary of response judging for unit "manson"

Your checklist	Items to be checked while in student mode			
	Check the displays: location of writing, sentence structure, clarity, spelling, etc.			
	Compare sequence of units to the flow diagram, figure 1.4.			
	Check all anticipated responses to see that PLATO interprets them as desired.			
	unit	concept of response	judgment	comment
	newton	apple	ok	(see figure 1.7)
		idea	ok	
		eve	no	
	wifehus	parent	ok	(see figure 1.12)
		child	no	
		adult	no	
	manson	adult	ok	(see figure 1.14)
		child	no	
		parent	no	
	Try some responses with capital letters, "extra words", and "ignorable words".			
	Try some obviously incorrect responses.			

Figure 1.15 Checklist for Exercise A

2. Variety in Displays and Sequencing

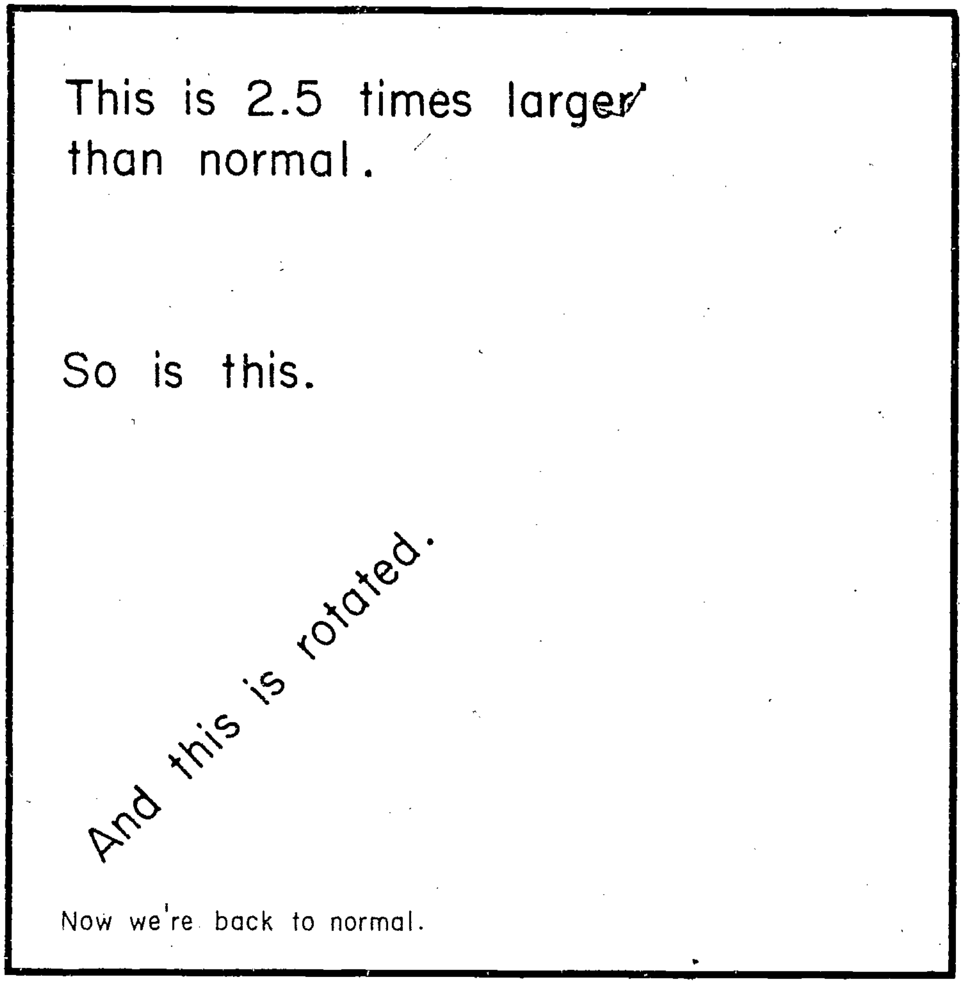
Sized Writing, Rotated Writing, and Drawing Lines

You can easily create eye-catching displays on the plasma panel using different sized characters. The command used for this is -size-. When normal size text is displayed, the whole character is plotted on the panel at once, but in "sized" writing each character is made from line segments drawn one line at a time. Sized writing is therefore slower than regular writing. If no -size- command is given, all writing is in regular size, called "size Ø". Sized writing may be placed at an angle by use of the -rotate- command. The tag of the -rotate- command is the number of degrees counter-clockwise from the horizontal that the writing is to be rotated. A -rotate- command only has effect on line-drawn characters, that is, when the size is other than Ø. The normal size characters (size Ø) are approximately the same size as line-drawn characters of size 1. The following unit illustrates the -size- and -rotate- commands.

```
unit    sizes
size    2.5
at      6Ø5
write   This is 2.5 times larger
at      1ØØ5
write   than normal.
at      14Ø5
write   So is this.
rotate  35
at      28Ø5
write   And this is rotated.
size    Ø
rotate  Ø
at      3ØØ5
write   Now we're back to normal.
```

Figure 2.1 shows how this unit appears on the student's panel. After a -size- command, all subsequent writing is in that size. The same holds true for -rotate-. Therefore you must return to normal writing by means of the following statements:

```
size    Ø
rotate  Ø
```



This is 2.5 times larger
than normal.

So is this.

And this is rotated.

Now we're back to normal.

Figure 2.1 Display created by unit "sizes"

Drawing lines and figures on the panel is done with the -draw- command. The unit below gives a definition of an acute angle, with the important words underlined. The underlining is the result of the -draw- command.

```
unit    acute
at      415
write   ANGLES
at      18Ø5
write   An acute angle is an angle that is less than 9Ø degr
        ees.
draw    18Ø8;1813;skip;184Ø;1844
```

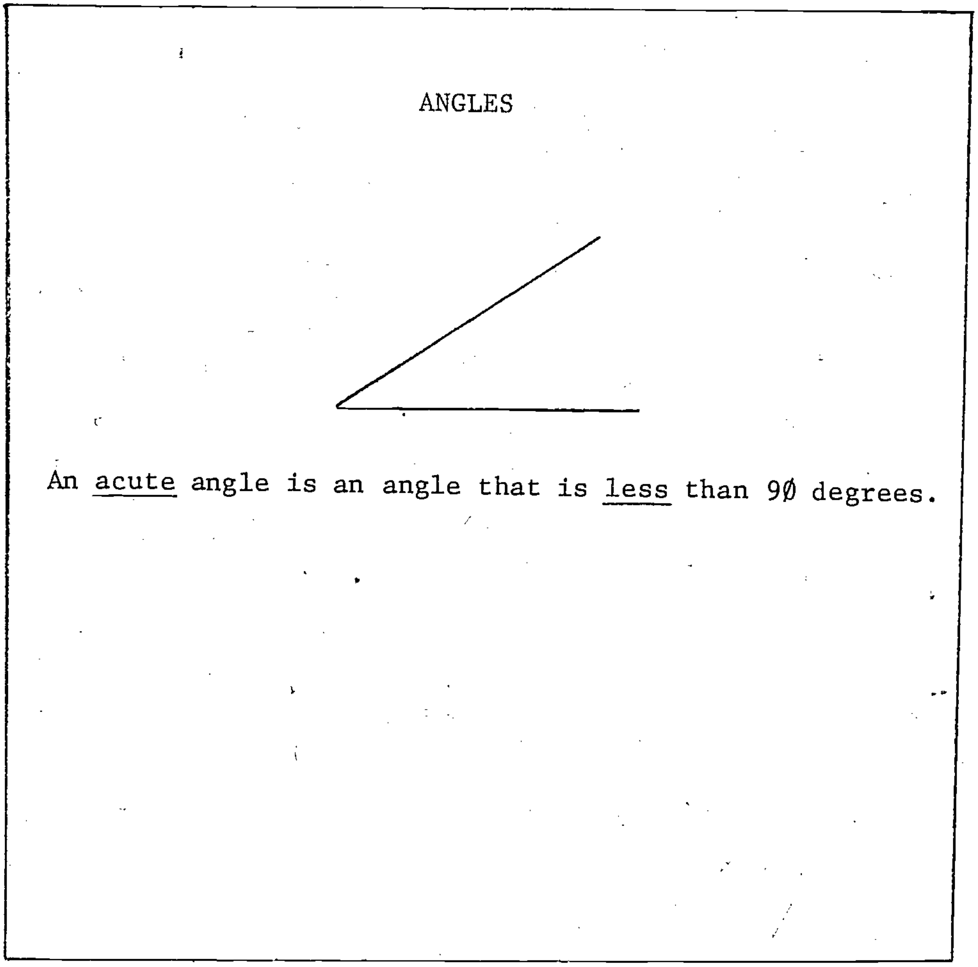
The -draw- statement draws lines under the words "acute" and "less". The word "acute" begins at position 18Ø8 and ends at 1813; "less" begins at 184Ø and ends at 1844. The tag of the -draw- command says where to draw from (18Ø8) and where to draw to (1813). The "skip" means to skip to the next point mentioned (184Ø) and draw from there to the last point given (1844). The positions are separated by semi-colons.

It would be very effective not only to define an acute angle but to illustrate it as well. This can be done by adding another -draw- command.

```
1    unit    acute
2    at      415
3    write   ANGLES
4    draw    938;1325;134Ø
5    at      18Ø5
6    write   An acute angle is an angle that is less than 9Ø degr
             ees.
7    draw    18Ø8;1813;skip;184Ø;1844
```

-draw- command on line 4 draws a line from position 938 to 1325 and from there to 134Ø. The unit now presents a display as shown in figure 2.2.

The individual entries of a TUTOR tag (such as the points specified in a -draw- tag) are called arguments. The arguments above are separated by semi-colons. Other separators are used in the tags of other TUTOR commands. Many TUTOR commands can have several arguments in their tags.



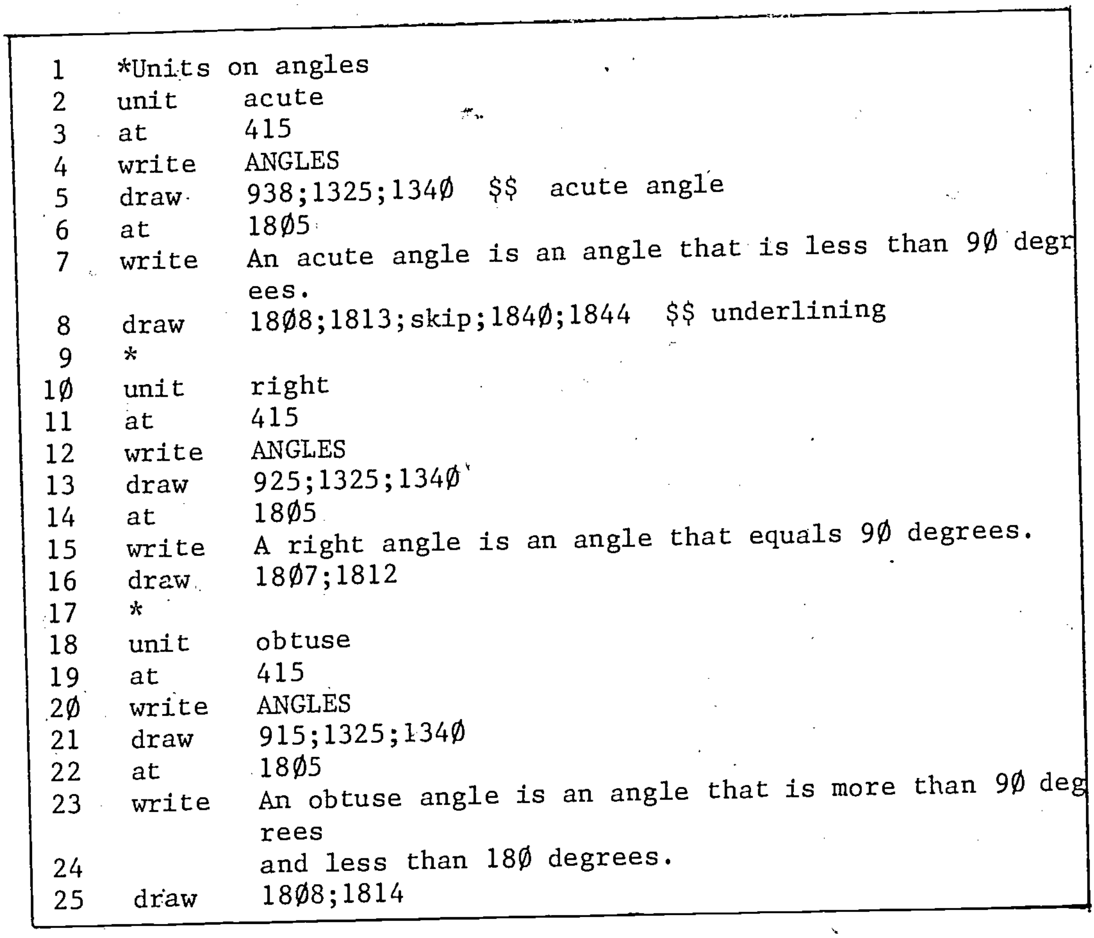
ANGLES

An acute angle is an angle that is less than 9Ø degrees.

Figure 2.2 Display produced by unit "acute"

Sequencing of TUTOR Units

So far units have been considered individually. In the context of a lesson, unit "acute" might be presented, followed by a definition and illustration of a right angle, and then a definition and illustration of an obtuse angle. In addition, the topic name "ANGLES" could be displayed in each unit. Figure 2.3 shows how this portion of the lesson might look. Lines beginning with the symbol * (lines 1, 9, and 17) are ignored by PLATO when the lesson is condensed for student use. You may use this symbol to insert comments to yourself and to make your lesson code more readable. Comments can also be added after the tag of any TUTOR statement by using the symbol $$ as on lines 5 and 8. Anything after $$ is ignored by PLATO.



```
*Units on angles
unit    acute
at      415
write   ANGLES
draw    938;1325;134Ø  $$ acute angle
at      18Ø5
write   An acute angle is an angle that is less than 9Ø degr
        ees.
draw    18Ø8;1813;skip;184Ø;1844  $$ underlining
*
unit    right
at      415
write   ANGLES
draw    925;1325;134Ø
at      18Ø5
write   A right angle is an angle that equals 9Ø degrees.
draw    18Ø7;1812
*
unit    obtuse
at      415
write   ANGLES
draw    915;1325;134Ø
at      18Ø5
write   An obtuse angle is an angle that is more than 9Ø deg
        rees
        and less than 18Ø degrees.
draw    18Ø8;1814
```

Figure 2.3 TUTOR code for units on angles

There are three units in this lesson, but how does the student progress from one to the next? The student starts in the first unit in the lesson. At the completion of a unit, PLATO waits for the student to press the NEXT key. When the student presses NEXT he proceeds to the unit physically following the one he is in, unless the author has specified otherwise. In the above lesson the student will first see unit "acute", then unit "right", then unit "obtuse", as illustrated in figure 2.4.

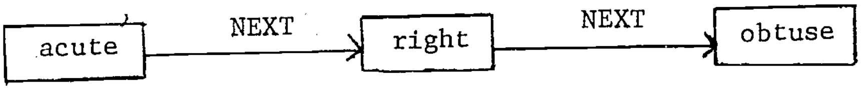


Figure 2.4 Sequencing of student through angle units

2.6

The statements

```
        at      415
        write   ANGLES
```

have to be included in each unit. Titles which will be used several times can be placed in a separate unit and that new unit can be "attached" to the units where it is needed by the -do- command.

A unit named "label" can be created:

```
        unit    label
        at      415
        write   ANGLES
```

and then the statements

```
        at      415
        write   ANGLES
```

can be replaced in each of the three units by the single statement

```
        do      label
```

The tag of the -do- command is the name of the unit to be attached. The contents of unit "label" will be treated as if inserted in the unit where the -do- statement occurs. The revised lesson is shown in figure 2.5

Since unit "label" has been inserted in the lesson, the physical order of the units no longer corresponds to the order in which they should be presented to the student. The lesson sequence after unit "acute" must be specified with a -next- command placed in unit "acute". The unit named in the tag of the -next- command is the unit which the student will proceed to when he presses NEXT after completing the current unit. If the current unit contains questions, it is completed only after all questions have been correctly answered. Figure 2.6 contrasts the order of the units in the written lesson with the order in which they appear on the student's panel.

```
unit    acute
next    right  $$ unit "right" is after "acute"
do      label  $$ attach "label"
draw    938;1325;134Ø
at      18Ø5
write   An acute angle is an angle that is less than 9Ø degr
        ees.
draw    18Ø9;1813;skip;184Ø;1844
*
unit    label  $$ attached unit
at      415
write   ANGLES
*
unit    right
do      label
draw    925;1325;134Ø
at      18Ø5
write   A right angle is an angle that equals 9Ø degrees.
draw    18Ø7;1812
*
unit    obtuse
do      label
draw    915;1325;134Ø
at      18Ø5
write   An obtuse angle is an angle that is more than 9Ø deg
        rees
        and less than 18Ø degrees.
draw    18Ø8;1814
```

Figure 2.5 Revised code for angle units

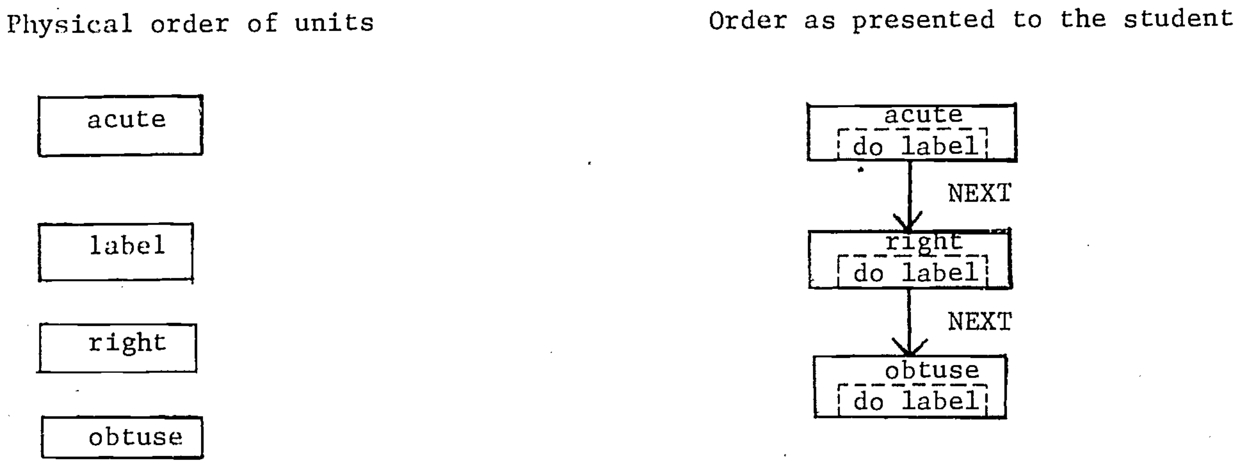


Figure 2.6 Comparison of unit order

2.8

Erasing Part of a Display

When the student leaves a unit and goes on to another, the entire panel is automatically erased. Sometimes it is desirable to remain in the same unit, erase part of the display, and add new information to the display. The commands -pause- and -erase- are useful in this context.

The -erase- tag may have either of two formats. The first type

```
at      1215
erase   3Ø
```

causes 3Ø characters to be erased, starting at line 12, character space 15. The second type

```
at      1215
erase   3Ø,3
```

causes the erasure of an area 3Ø characters by 3 lines. Lines 12, 13, and 14 will be erased starting at the fifteenth space and continuing through the fcrty-fourth space.

The -pause- command has several forms, one of which has no tag. When a -pause- with no tag is encountered, PLATO stops processing and waits for the student to press any key. When a key is pressed, the rest of the unit is processed. Unit "compang" below uses a -pause- command with no tag.

```
unit    compang
at      1Ø1Ø
write   Complementary angles are two angles whose sum
        equals 9Ø degrees.
pause
at      141Ø
write   Supplementary angles are two angles whose sum
        equals 18Ø degrees.
```

The first sentence will be written at 1Ø1Ø, and then nothing will happen until the student presses a key. When he does, the second sentence will be written at 141Ø.

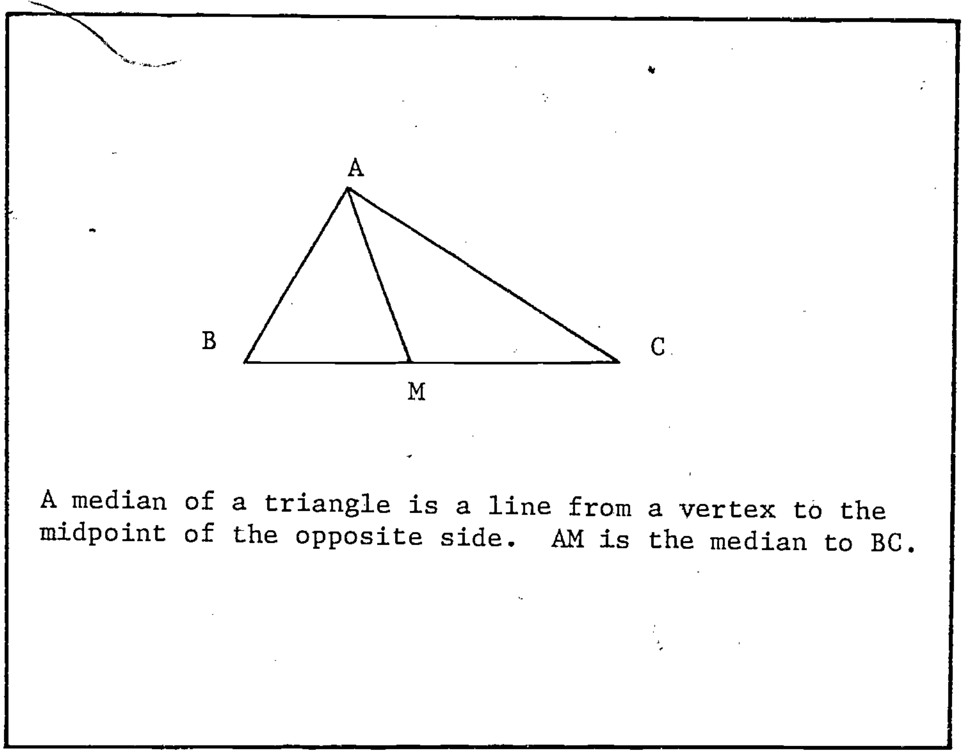
Figure 2.7 is a unit containing a rather complicated drawing. There are two pieces of information about the drawing to be presented to the student. The -pause- and -erase- commands are used to present one piece of information, erase it, then present the next piece of information.

The first 13 lines of the unit draw and label a figure and put some text

```
unit    median
draw    1325;928;134Ø;1325;skip;1332;928
at      928
write   A
at      1324
write   B
at      134Ø
write   C
at      1431
write   M
at      18Ø5
write   A median of a triangle is a line from a vertex to th
        e
        midpoint of the opposite side.  AM is the median to
        BC.
pause
*Wait for student to press a key
*
at      18Ø5
erase   59,2
draw    932;1432
at      932
write   P
at      18Ø5
write   A perpendicular bisector of a side of a triangle is
        a line
        that bisects and is perpendicular to a side.  PM is
        the
        perpendicular bisector of BC.
```

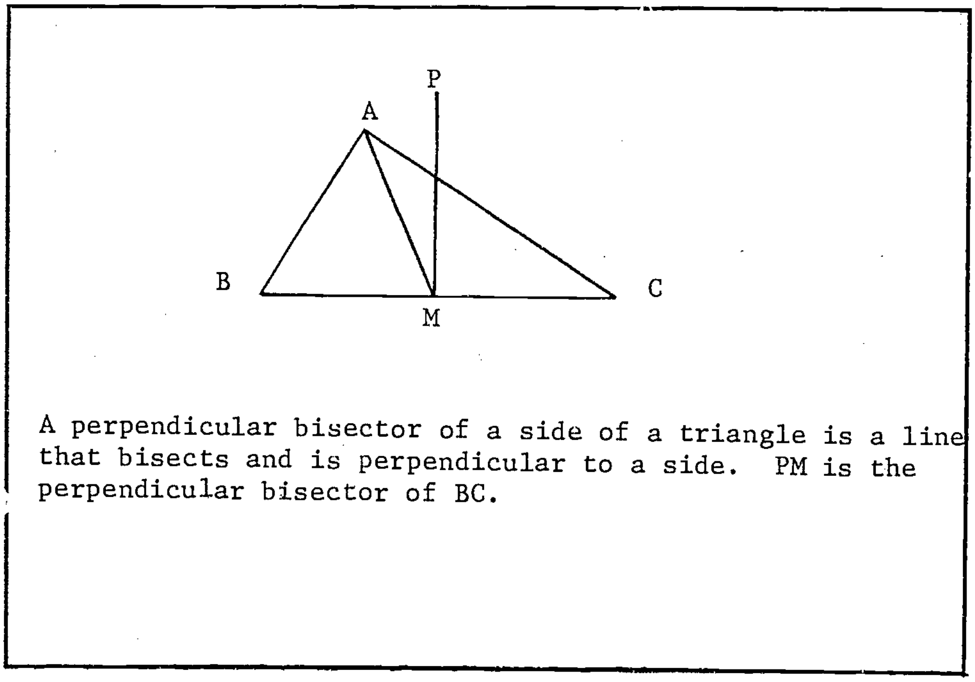
Figure 2.7 Code for unit "median"

on the panel. Figure 2.8 shows how this part of the unit appears on the student's panel. Then the -pause- command causes PLATO to wait for the student to press a key. When a key is pressed the first definition is erased, a new line is added to the drawing, and the second definition is written in the place of the first. Figure 2.9 shows how the unit looks to the student after this is done. When large amounts of text are to be displayed, the -pause- command with no tag allows the student to read one portion of the text before another is displayed.



A median of a triangle is a line from a vertex to the midpoint of the opposite side. AM is the median to BC.

Figure 2.8 Initial display of unit "median" (before the -pause-)



A perpendicular bisector of a side of a triangle is a line that bisects and is perpendicular to a side. PM is the perpendicular bisector of BC.

Figure 2.9 Final display of unit "median" (after the -pause-)

Displaying Circles

In addition to line segments, circles can be drawn on the panel. However, to specify the size and location of the circle, the coarse grid is not used. You must use the more precise positioning scheme called the fine grid. The fine grid is 512 dots by 512 dots. The dots are numbered from Ø to 511, figure 2.10, with 6Ø dots to the inch. Any screen dot may be referenced in fine grid by giving its x-axis and y-axis location. Illustrated in figure 2.11 is position 7Ø,21Ø. It is 7Ø dots from the left and 21Ø dots from the bottom of the panel.

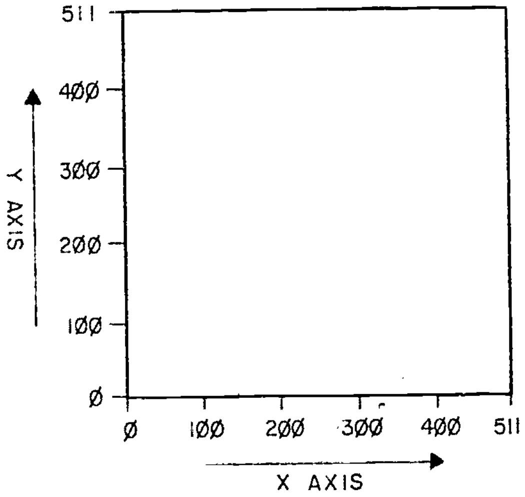


Figure 2.10 Fine grid positioning scheme

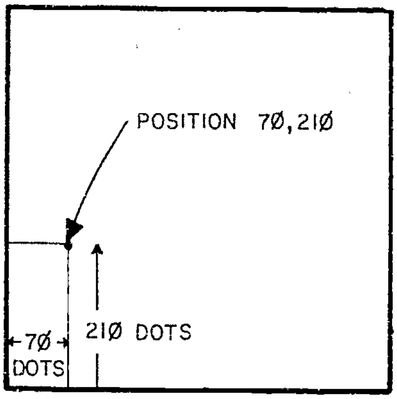


Figure 2.11 Fine grid position: 7Ø,21Ø

To draw a circle you must specify the radius in number of fine grid dots and the fine grid location of the center. The statement

```
circle  4Ø,126,338
```

draws a circle with a radius of 4Ø dots. The center of the circle is 126 dots from the left and 338 dots from the bottom of the panel. A dashed or broken circle may be drawn with the -circleb- command. The statement

```
circleb 4Ø,226,338
```

draws a broken circle with a radius of 4Ø dots, the center of which is at fine grid location 226,338.

Partial circles and partial broken circles can be created by including in the tag the initial and final angles of the partial circle. The positive x-axis is taken to be Ø° and the positive y-axis is 9Ø°.

```
        circle 4Ø,326,338,Ø,18Ø
radius   x and y locations of center   initial angle   final angle
```

These three commands are illustrated in figure 2.12. From left to right they are the 3-argument -circle- command, the -circleb- command, and the 5-argument -circle- command (the partial circle).

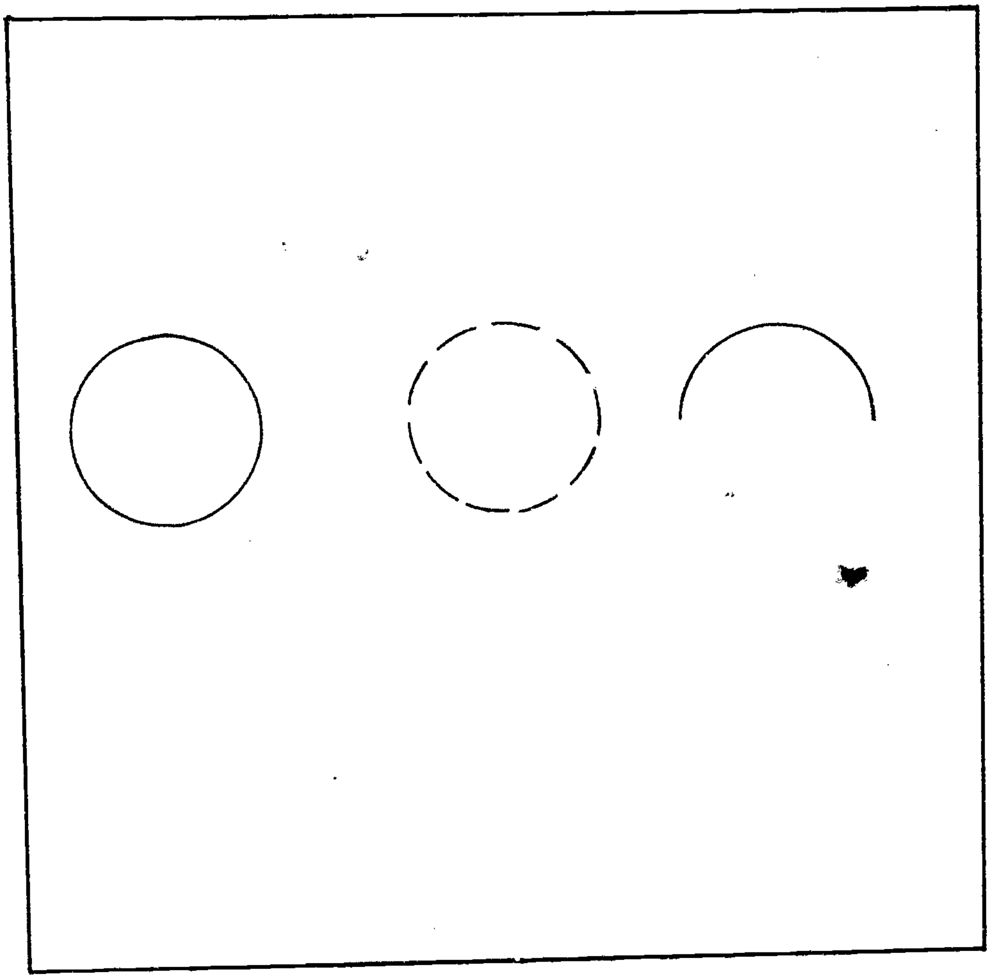


Figure 2.12 Some circles

2.14

You can use fine grid coordinates with the -at- command to position writing more precisely. The format for the fine grid -at- tag is: x-location,y-location. A comma separates the two fine grid coordinates. Below is a unit that draws a circle and writes the words "a circle" inside the figure.

```
unit    circle
circle  4Ø,126,338
at      94,338
write   a circle
```

Fine grid locations can also be part of the tag of a -draw- command. Each location is set off by a semicolon. If a location is specified in fine grid, the x and y coordinates are separated by a comma.

```
draw     2213;4Ø,1ØØ;312,12Ø;2213
          /     \/      \/     \
     coarse   fine     fine   coarse
```

Any coarse grid location may also be given in fine grid because a coarse grid location is actually an area 8 fine grid dots wide by 16 grid dots high. Tags of -at-, -draw-, and -arrow- commands, whether in coarse or fine grid, and tags of -circle- and -circleb- commands may contain arithmetic expressions.

Exercise B - General Comments

The sequencing of a student through the units in exercise B is outlined in the flow diagram, figure 2.13. Attached units are indicated by circles and are attached to main units by dotted lines. Enter sample lesson B through the index of lesson "introtutor" and use the flow diagram as a check that you have tried all the units. As you study the sample lesson, try to anticipate what TUTOR commands you will need.

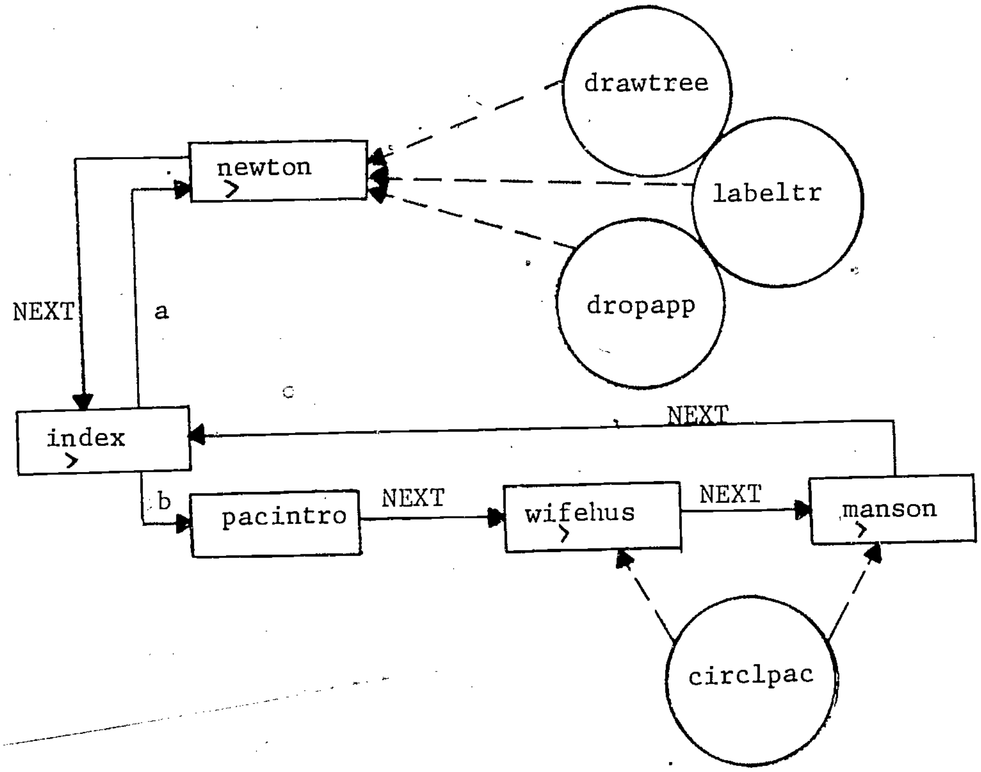


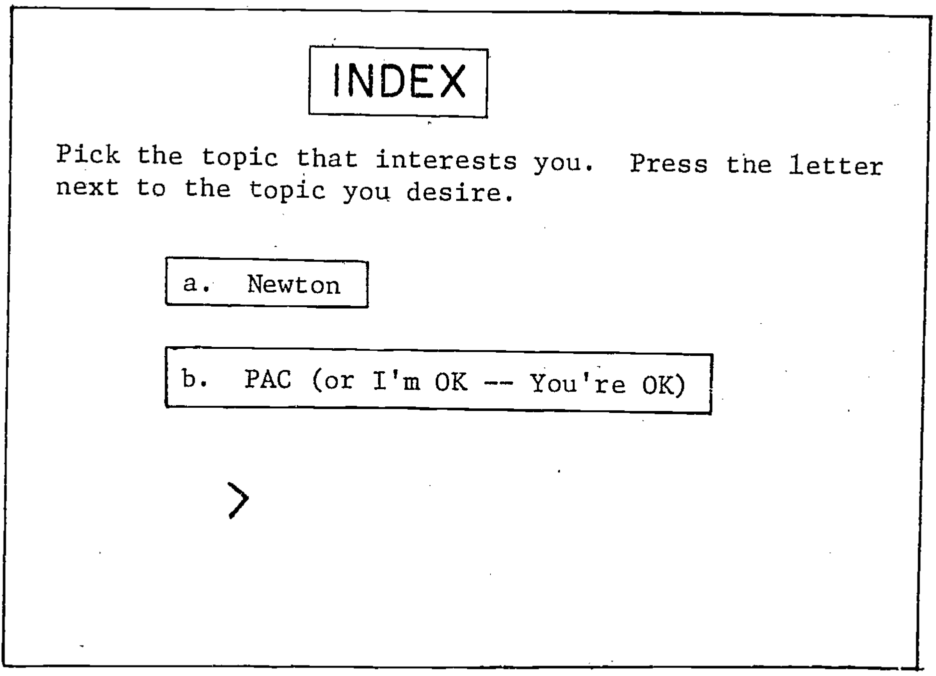
Figure 2.13 Flow diagram for Exercise B

The Index

From the author mode display, enter your lesson and change the name of block "a" to "index". To do this enter block "a", press the LAB key, then follow directions. Make sure you press NEXT after typing "index" or PLATO will not remember the name change.

Create a unit "index" in block "index" so it resembles figure 2.14. When your index display suits you, you are ready to insert the commands to handle student responses. The sequence of commands you will need is similar to what you used in unit "newton":

```
unit     index
*
*TUTOR commands to produce display
*
arrow    81Ø  $$ your tag probably will be different
specs    bumpshift
answer   a
*commands done only after a response of "a"
answer   b
*commands done only after a response of "b"
```



INDEX

Pick the topic that interests you. Press the letter next to the topic you desire.

a. Newton

b. PAC (or I'm OK -- You're OK)

>

Figure 2.14 Display from unit "index"

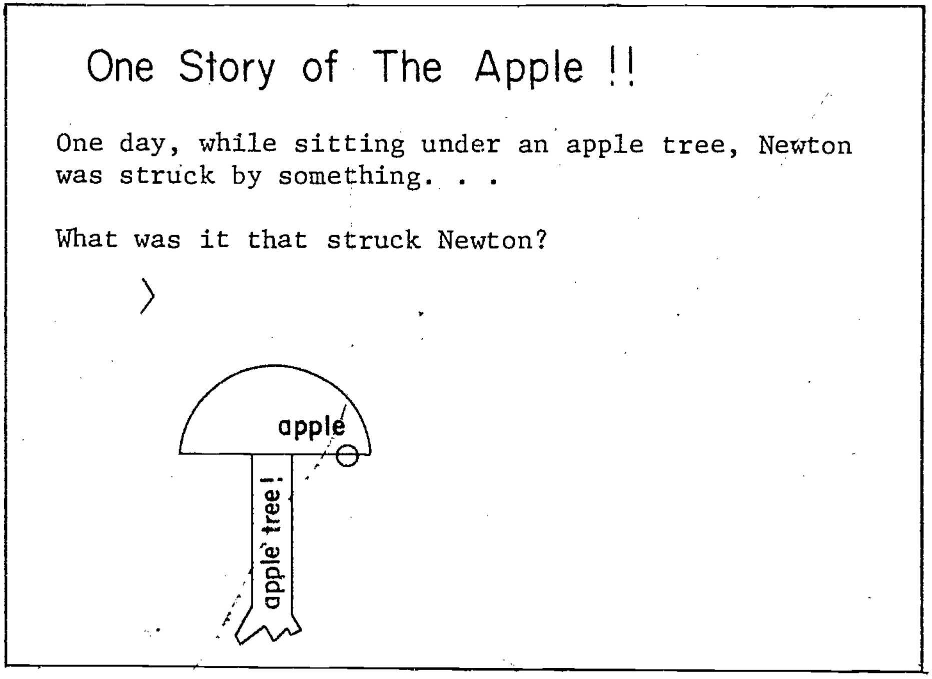
In exercise A you saw that commands after an -answer- or -wrong- command and before another -answer- or -wrong- command are done only if the student's response matches that particular -answer- or -wrong- command. To branch according to a student's response, you will need a -next- command after each -answer- command. The tags of the -next- commands will be the name of the unit the student should see next when his response matches the preceding -answer- command. Insert the necessary commands so your unit "index" branches the student appropriately and then try your lesson in student mode.

Your lesson sequencing is not yet like that shown in the flow diagram since the student is not returned to unit "index" after completing the topic he chose. To return the student to unit "index" you will need -next index- statements in the last units of both topics. These -next index- statements should be placed immediately after the -unit- command. Condense your lesson and check the new sequencing.

Physics Units

The display changes in unit "newton" require using sized writing, drawing a tree, labeling the tree, and dropping an apple. The last three require attaching units with -do- commands, because these units will be attached in several different places later in the exercises. The attached units should be placed in a new block in your lesson entitled "do newton".

One of the attached units should draw a tree resembling the tree in figure 2.15. Call this unit "drawtree". Your tree need not be identical to the sample lesson's tree. Concentrate on becoming familiar with locations on the panel rather than duplicating the sample lesson's tree. The second attached unit, "labeltr", should write "apple tree!" on the trunk and "apple" in the branches. The last unit in block "do newton", named unit "dropapp", will drop the apple from the tree. To do this you will need a series of commands similar to those in figure 2.16. Each -circle-, -at-, and -erase- command will move the apple once. The last command in the unit should be -circle- so the apple is not erased from the panel.



One Story of The Apple !!

One day, while sitting under an apple tree, Newton was struck by something. . .

What was it that struck Newton?

〉

Figure 2.15 Display for unit "newton"

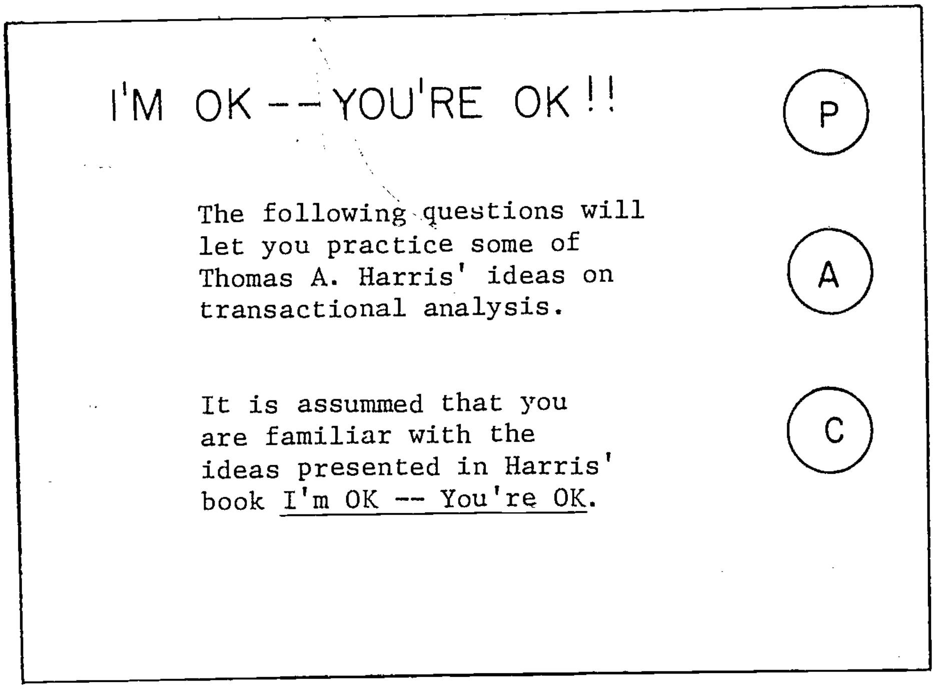
```
unit    dropapp  $$ unit to drop apple
circle  4,178,224
at      172,22Ø
erase   2
circle  4,178,223
at      172,219
erase   2
circle  4,178,219
at      172,215
erase   2
circle  4,178,21Ø
at      172,2Ø6
erase   2
circle  4,178,194
at      172,19Ø
erase   2
circle  4,178,169
at      172,165
erase   2
circle  4,178,133
at      172,129
erase   2
circle  4,178,84
```

Figure 2.16 Unit "dropapp" drops the apple

Since the labeled tree should appear on the panel before the student is asked the question, the corresponding -do- commands should be inserted before the -arrow- command. The apple should drop from the tree only after the student answers correctly, so the appropriate -do- command should be inserted after each -answer- command.

Psychology Units

The display changes in unit "pacintro", as seen in figure 2.17, require using sized writing, underlining with a -draw- command, and adding three circles. To position the letters P A C in the centers of the circles, you may need to use fine grid tags with the -at- commands. There is an editing directive named "id" (insert a display) which facilitates the creation of displays. To try this press "id" from the line listing display, then NEXT. Help is available while using the "id" option by pressing the HELP key.



I'M OK -- YOU'RE OK!!

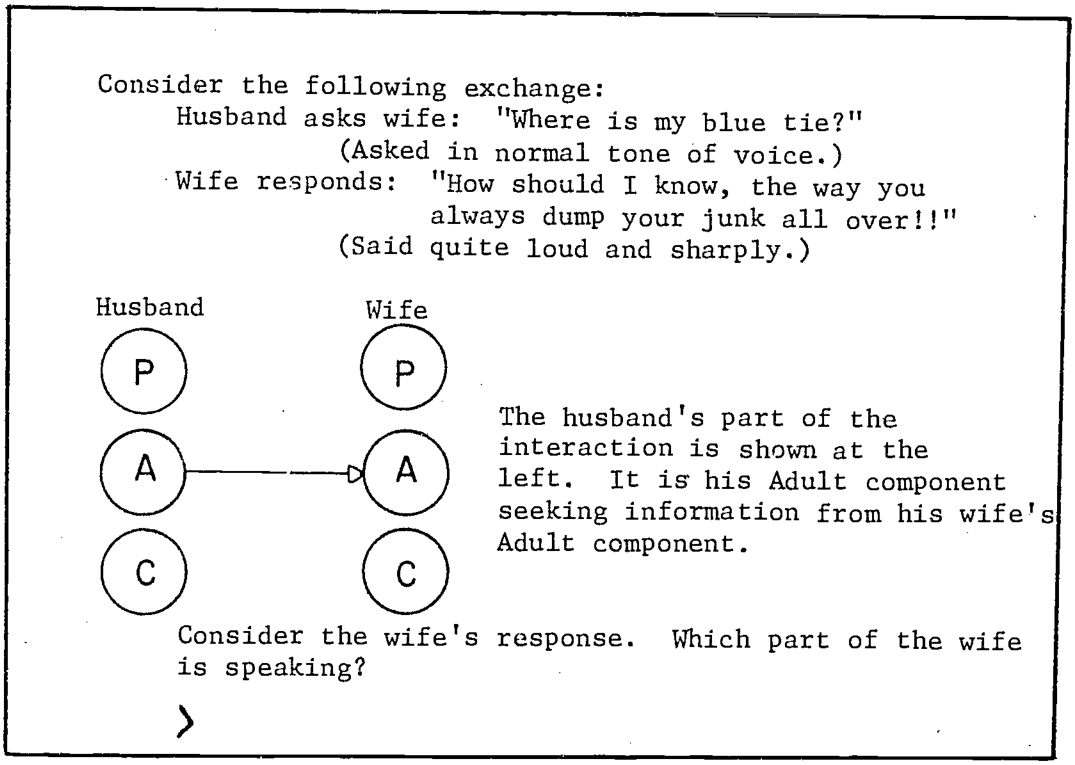
The following questions will let you practice some of Thomas A. Harris' ideas on transactional analysis.

It is assummed that you are familiar with the ideas presented in Harris' book I'm OK -- You're OK.

Figure 2.17 Display from unit "pacintro"

Both of the units which depict interactions (see figures 2.18 and 2.19) will require the same types of changes. Circles and lines are needed for the graphic representation of the interactions. Since the same circles are used in units "wifehus" and "manson", a new unit should be created, named "circlpac", which can be attached by -do- commands. The lines are not the same in the two units, so different -draw- commands are required.

To complete this exercise, refer to the checklist in figure 2.2∅ while using your lesson in student mode.



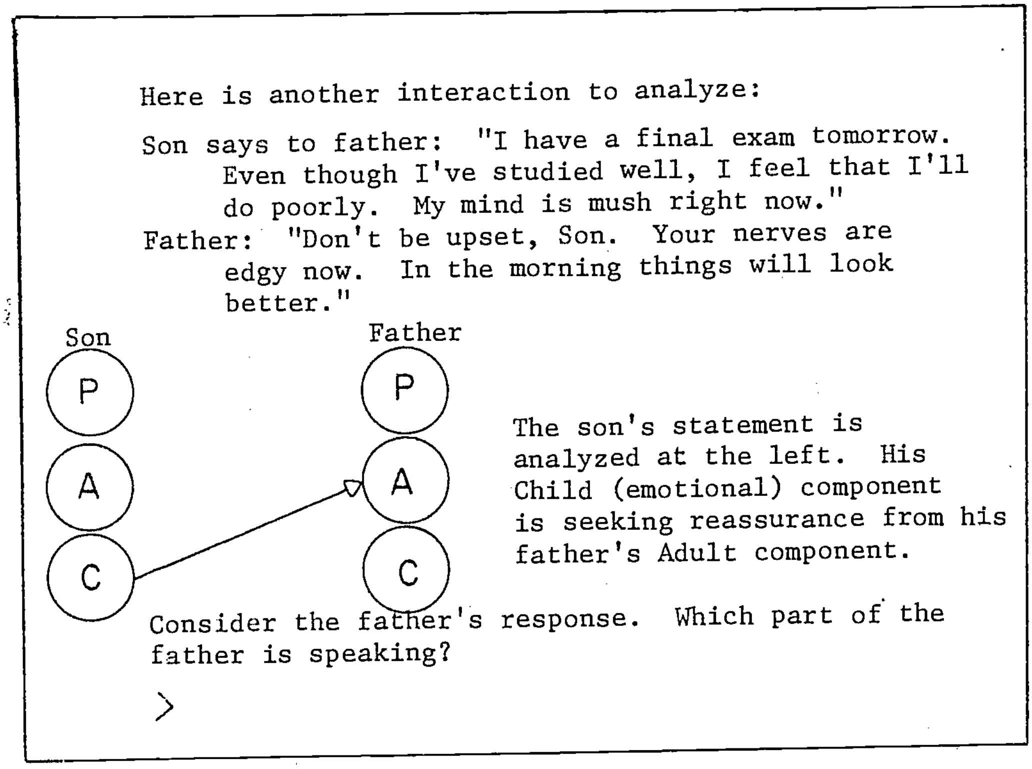
Consider the following exchange:
Husband asks wife: "Where is my blue tie?"
(Asked in normal tone of voice.)
Wife responds: "How should I know, the way you
always dump your junk all over!!"
(Said quite loud and sharply.)

The husband's part of the interaction is shown at the left. It is his Adult component seeking information from his wife's Adult component.

Consider the wife's response. Which part of the wife is speaking?

>

Figure 2.18 Display from unit "wifehus"



Here is another interaction to analyze:

Son says to father: "I have a final exam tomorrow. Even though I've studied well, I feel that I'll do poorly. My mind is mush right now."
Father: "Don't be upset, Son. Your nerves are edgy now. In the morning things will look better."

The son's statement is analyzed at the left. His Child (emotional) component is seeking reassurance from his father's Adult component.

Consider the father's response. Which part of the father is speaking?

>

Figure 2.19 Display from unit "manson"

your checklist	Items to be checked while in student mode
	Compare the sequence of units to the flow diagram, figure 2.13.
	Check the displays: Does your unit "index" resemble figure 2.14?
	Does your unit "newton" resemble figure 2.15?
	Does your unit "pacintro" resemble figure 2.17?
	Does your unit "wifehus" resemble figure 2.18?
	Does your unit "manson" resemble figure 2.19?
	Check the animation: Does the apple drop after a correct response?
	Is the old apple erased?
	Does the last apple remain on the panel?

Figure 2.2Ø Checklist for Exercise B

3. Doing Calculations and Using Variables

Student Variables

In order to individualize a lesson, write lessons efficiently, and make use of the power of the PLATO computer, you will need to store and use numeric information which may change in value as the student proceeds through the lesson. For example, you may wish to keep a record of how many errors a student makes so that you can have your lesson give the student some review after a given number of errors. This sort of information can be kept in storage locations within the computer called student variables. When a student is registered on PLATO, 150 student variables are reserved for him. These student variables are unimaginatively called v1, v2, v3...v148, v149, v15Ø.

These variables are called student variables because each of the many students who may simultaneously be studying your lesson has his own private set of 150 variables. You might use variable v23 to count the number of correct responses on a certain topic, which will be different for each student. If there are forty students working on your lesson, TUTOR is keeping track of forty different "v23's", one for each student. This is done automatically for you, so that you can write the lesson with one individual student in mind, and v23 may be considered simply as that individual student's count of correct responses. Thus one student might be sent to a remedial unit because his variable 23 shows that he did poorly on this topic. Another student might be jumped ahead because her variable 23 indicates an excellent grasp of the material. It is through manipulation of the student variables that a lesson can be highly individualized for each student.

The -calc- command is used to store a number in a variable or to change the number in a variable. The statement

```
calc    v22⇐6.3
```

causes the value 6.3 to be placed in the twenty-second variable. The above -calc- statement is read, "Assign the value 6.3 to the twenty-second variable."

The statement

```
        calc    v16←v12
```

causes the value stored in variable 12 to be assigned to variable 16 (the value in v12 is not changed). Not only constants and variables, but arithmetic expressions can be used in -calc- statements. Upon encountering the statement

```
        calc    v31←14+2×6
```

PLATO evaluates the expression on the right of the assignment arrow (←) and stores the result, 26, in v31. The order of operations in TUTOR is exponentiation, multiplication, division, addition and subtraction. Addition and subtraction are done in order of occurence, from left to right. Operations inside parentheses are done before the whole expression is evaluated, so while 14+2×6 gives a result of 26, the expression (14+2)×6 gives the result 96. The arithmetic symbols and the assignment arrow are on the black keys on the left of the keyset. Figure 3.1 summarizes the order of arithmetic operations in TUTOR and figure 3.2 illustrates some calculations in TUTOR.

A specialized TUTOR calculational command is -zero-, which assigns the value Ø to a variable. The statements

```
        calc    v83←Ø
        zero    v83
```

are equivalent. If you need to set many consecutive variables to Ø, use

```
        zero    v7,3Ø
```

This puts the value Ø in 3Ø consecutive variables, starting at v7. The variables v7, v8 . . . through v36 will all contain Ø.

		Operation	Examples		
	First	exponentiation	$v12^3$ (use SUPER key)		
Order of Operations	↓	multiplication	v21×4	or	v21*4
		division	1ØØ÷v17	or	1ØØ/v17
	Last	addition, subtraction	v42+31		
			1Ø43-v75		

Figure 3.1 Order of arithmetic operations in TUTOR

-calc-	statement	value of v43	value of v6
calc	v6←Ø	----	Ø
calc	v6←11	----	11
calc	v43←1Ø^{3}	1ØØØ	11
calc	v6←1+v6	1ØØØ	12
calc	v6←4×v6	1ØØØ	48
calc	v43←v43-17Ø	83Ø	48
calc	v43←v43÷1ØØ	8.3	48
calc	v43←v6+v43*1Ø	131	48
calc	v6←(25+v6÷8-v43)/1Ø	131	-1Ø
calc	v43←((v6)^{2}+6×v6)/5	8	-1Ø

Figure 3.2 Some calculations in TUTOR

Giving Variables Meaningful Names

TUTOR statements containing student variables will be more readable if the variables are given meaningful names. You can create your own names with the -define- command:

```
define  wrongs=v1,percent=v2,radius=v3
```

Throughout your lesson now you can use "wrongs" in place of "v1", "percent" instead of "v2", and "radius" in place of "v3". The names must be 7 or fewer characters and may not begin with a number or contain a mathematical operator. If defined names are not used it will be difficult to remember that v3 contains the radius and not, say, the percent.

The -define- statement tells TUTOR how to interpret the descriptive names when they are encountered in other statements. The -define- statement must precede the first reference to any of the defined names. It is convenient to place the -define- command in what is called the initial entry unit, or i.e.u. The i.e.u. consists of commands at the beginning of the lesson which precede the first -unit- command. The i.e.u. is automatically executed every time someone enters the lesson.

If you wish you may give a name to the set of defined names:

```
define  setname,wrongs=v1,percent=v2,radius=v3
```

The name of the set (like the names of variables) must be 7 or fewer characters and may not begin with a number or contain an operator.

Through the use of variables you can make your lesson quite flexible. The following units use variables to draw a circle at different points on the display panel.

```
 1   define  mynames,radius=v1,centerx=v2,centery=v3
 2   unit    drawit  $$ the i.e.u. is above this unit statement
 3   calc    radius←2Ø
 4           centerx←35Ø
 5           centery←8Ø
 6   do      circles
 7   calc    centerx←2ØØ
 8           centery←185
 9   do      circles
1Ø   calc    radius←8Ø
11   do      circles
12   *
13   unit    circles
14   circle  radius,centerx,centery
```

On lines 3 - 5, the initial values are stored in the three variables. If a -calc- statement is continued for more than one line, you do not have to repeat the command. Unit "circles" is attached (line 6), drawing a circle with a radius of 2Ø dots. Its center is at fine grid location 35Ø,8Ø. Lines 7 and 8 set up a new center by putting new values in variables "centerx" and "centery" in place of the old ones. When unit "circles" is attached again (line 9), the new circle is the same size (radius), but its center is at a different place: 2ØØ,185. Line 1Ø puts the new value 8Ø in "radius". The variables "centerx" and "centery" have not been changed; they still contain the values 2ØØ and 185 respectively. When line 11 is executed, the resulting circle will have the same center as the second circle, but a larger radius.

Displaying Values of Variables

You can display on the panel the value of any variable by using the -show- command. The -show- command displays the value with no spaces preceding or following the number. Other commands display the values of variables in different formats. Unit "showing" evaluates an expression, stores the result in a variable, and displays it.

```
define  area=v67
unit    showing
calc    area⇐3.14×4²
at      12Ø5
show    area
```

If you want to display a value in the middle of some text, you may do it either of two ways. In unit "ashow" a space must be left after the word "is" in the first -write- tag and before "square" in the second -write- tag.

```
unit    show
calc    area⇐3.14×4²
at      12Ø5
write   The area is  $$ space here for nice display
show    area
write    square feet.  $$ space before "square"

unit    bshow
calc    area⇐3.14×4²
at      12Ø5
write   The area is ⟨s,area⟩ square feet
```

These two units do exactly the same thing. The ⟨ ⟩ are called "embed" symbols, because they allow you to embed the -show- command in the tag of the -write-. To type the embed symbols, strike the shifted ▭ key, then Ø for ⟨ or the shifted ▭ and 1 for the ⟩ . The shifted ▭ key is called ACCESS. It is to the right of the HELP key. The first argument within the embed symbols stands for which show-type command you want (s for -show-, t for -showt-) and the second is the name of the variable to be shown.

Displaying Tables of Numbers

The -show- command displays numbers with no leading spaces so a table of numbers will be aligned with some left margin if a -show- is used. However the standard table format has the entries aligned along a right margin. The -showt- command will display a set of numbers with this format. The statement

```
        showt   v7
```

will display the current value of v7. The location of writing by a -showt- command is determined by the preceding -at-. The statements

```
        calc    radius⇐21    $$ "radius" is defined
        at      1817
        showt   2×πradius
```

will display a circumference value of 131.947 starting at line 18 and character 17 (2π×21=131.947).

When displaying a table of information, it is often convenient to use an expression in the tag of an -at- command which is a function of some initial location in a table. The expression can be incremented by 1ØØ after each line of data in a table is displayed so the following line of data is placed in an appropriate location.

The units in figure 3.3 display several circles and a table of radii and circumferences; refer to figure 3.4. The locations for the -showt- commands are a function of a point of reference (line 12) and are incremented by 1ØØ (line 3Ø) after each line of data is displayed. The location of the circumference (2π×radius) is always 2Ø characters to the right of radius (line 26).

```
* "place" and "radius" are defined in the i.e.u.
unit    table
at      4Ø7
write   Table of radii and circumferences.  Press NEXT for
        each entry to be displayed
draw    1224;2424;2462;1262;1224;skip;1424;1462;skip;1243;
        2443
at      23Ø,296
write   radius
at      36Ø,296
write   circumference
calc    place←1628  $$ point of reference
        radius←1  $$ set initial radius
do      circ
do      circ
do      circ  $$ 8 table entries, 1 for each -do-
do      circ
do      circ
do      circ
do      circ
do      circ
*
unit    circ
pause
circle  radius,9Ø,23Ø
at      place
showt   radius
at      place+2Ø
showt   2π×radius
calc    place←place+1ØØ  $$ move down 1 line
calc    radius←radius+1Ø  $$ increase the radius
```

Figure 3.3 TUTOR code for radii and circumferences table

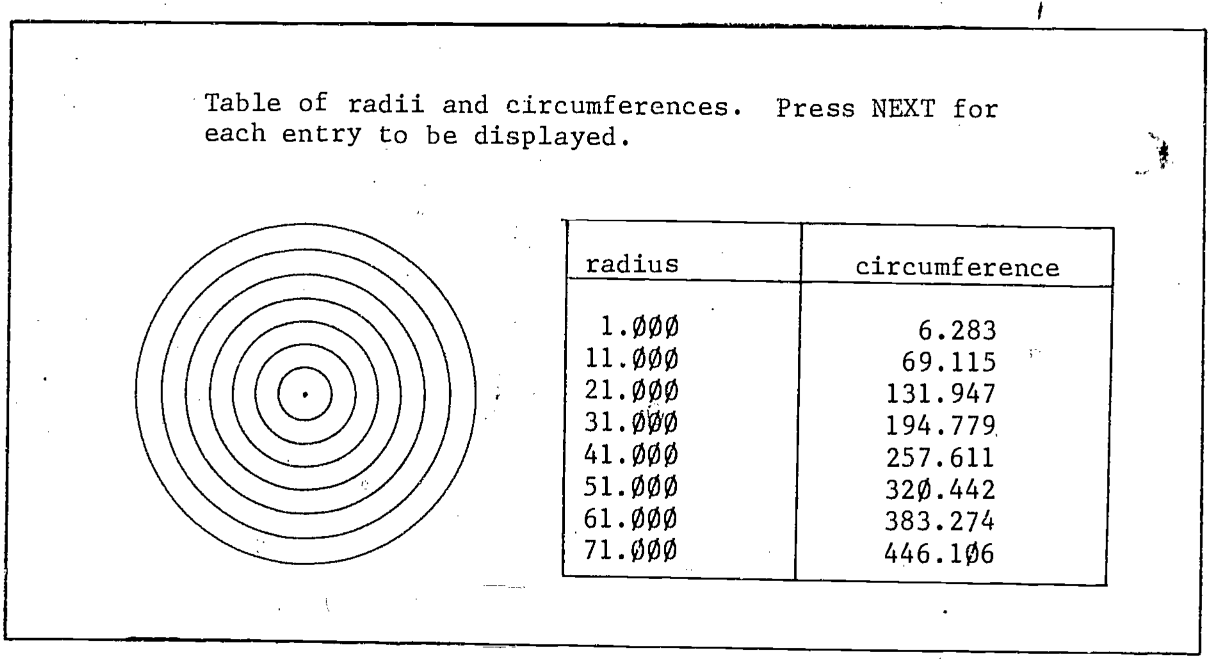


Table of radii and circumferences. Press NEXT for each entry to be displayed.

radius	circumference
1.ØØØ	6.283
11.ØØØ	69.115
21.ØØØ	131.947
31.ØØØ	194.779
41.ØØØ	257.611
51.ØØØ	32Ø.442
61.ØØØ	383.274
71.ØØØ	446.1Ø6

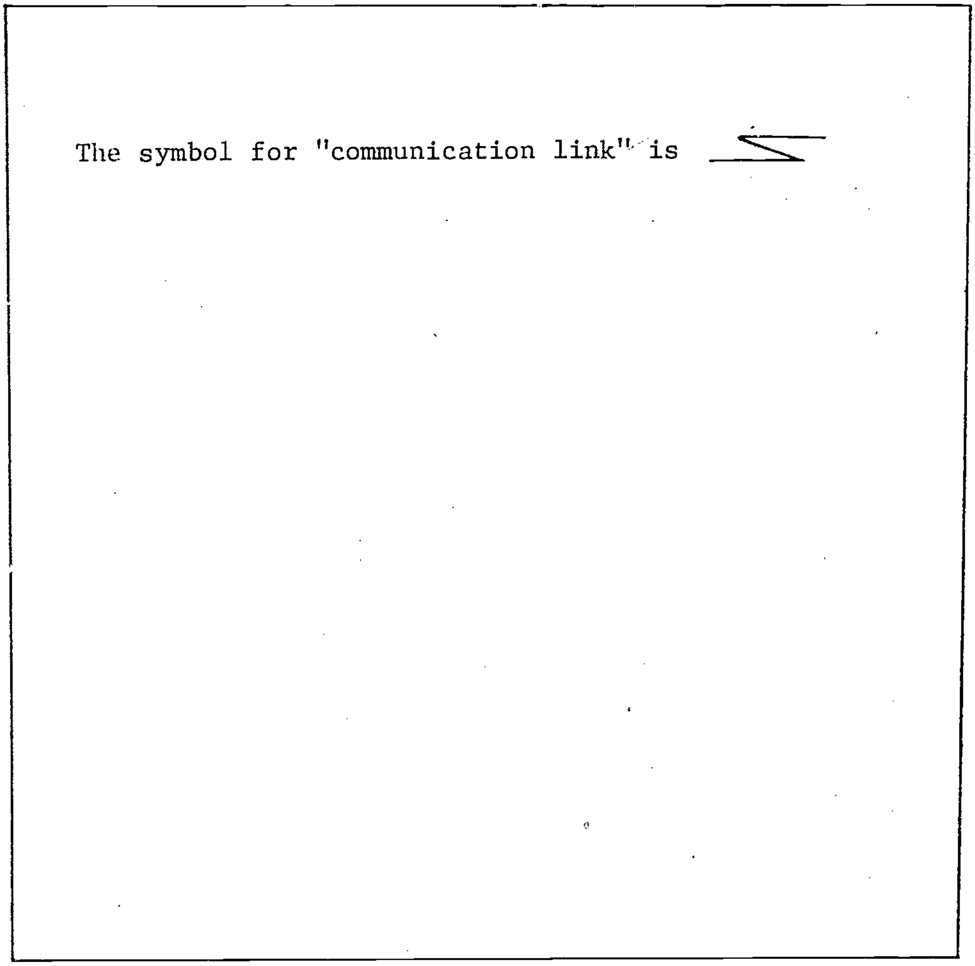
Figure 3.4 Student display of radii and circumferences table

Systems Reserved Words

In addition to student variables, there are about 40 "systems reserved words". These are variables, like student variables in that each student has his own set. You may show their values and use them in expressions, but you may not change these values. The systems reserved words contain information authors frequently need. One of the reserved words is "ntries"; it contains the number of tries a student has made on a question (-arrow-). Another very useful systems reserved word is "where", which contains the coarse grid location of the last panel activity. The following unit shows how "where" can be used.

```
unit    symbol
at      1Ø1Ø
write   The symbol for "communication link" is
draw    where+2;where+7;(where-1ØØ)+5;(where-1ØØ)+1Ø
```

Figure 3.5 shows how this unit looks on the student's panel. The -draw- command draws a figure composed of three line segments. The first is 5 characters long and begins 2 spaces to the right of the word "is" on line 1Ø. The next segment is a diagonal line running from the end of the first line segment up to the ninth line, determined by the expression (where-1ØØ), at 5 character spaces to the right of the word "is"; and the last segment is a straight line from this previous position to the tenth character space to the right of the word "is". The symbol is drawn right after the sentence, and by using "where" you do not need to count all the characters in the line to determine where to place the drawing. The systems reserved words "wherex" and "wherey" contain the x and y positions in fine grid of the last writing or drawing. They may be used like "where" when you are using fine grid.



The symbol for "communication link" is

Figure 3.5 Display from unit "symbol"

Exercise C - General Comments

The physics topic has been expanded to include a table which illustrates the relationship between time and distance as an apple falls to the ground. The order of the physics units is described in the flow diagram, figure 3.6. In this and subsequent exercises, attached units are not shown on the flow diagrams. Use sample lesson C and try these units.

The Initial Entry Unit

In this exercise, the defined variables "radius", "xcenter", "ycenter", "time", "locate", and "pacerr" are used. To insert the -define- command on the first line of block "index" (before the -unit index- statement) use "i∅" as the editing directive. This places the -define- statement ahead of the first -unit- command, creating an initial entry unit. You should use consecutive variables (e.g. v1 through v6) in the tag of the -define- statement. This allows you to tell at a glance which of the 15∅ student variables you have already used.

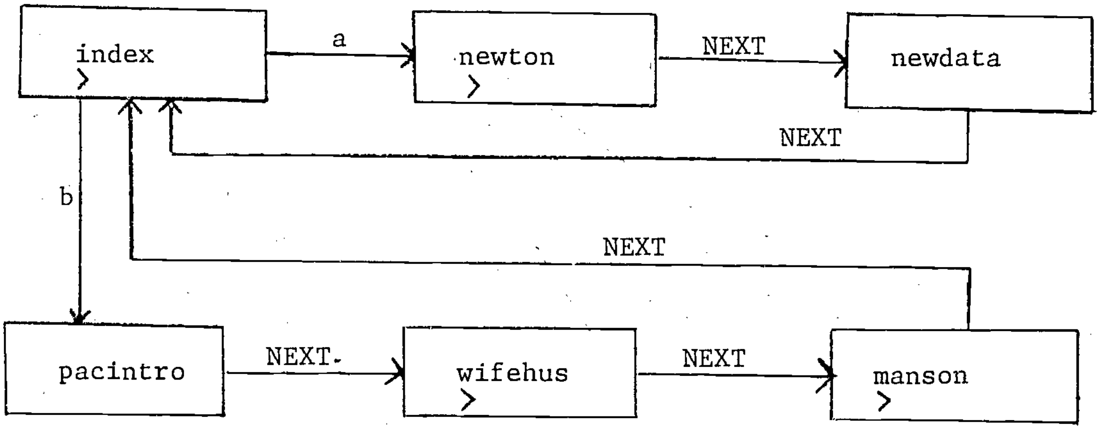


Figure 3.6 Flow diagram for Exercise C

Physics Units

When the apple was dropped in the previous exercises, a non-mathematical approach was used. However, Newton discovered that the distance an object falls is directly related to some constant force (gravity) multiplied by the square of the time. Therefore, the y-coordinate (distance) will decrease as a function of the time squared. Unit "dropapp" is to be rewritten using this principle.

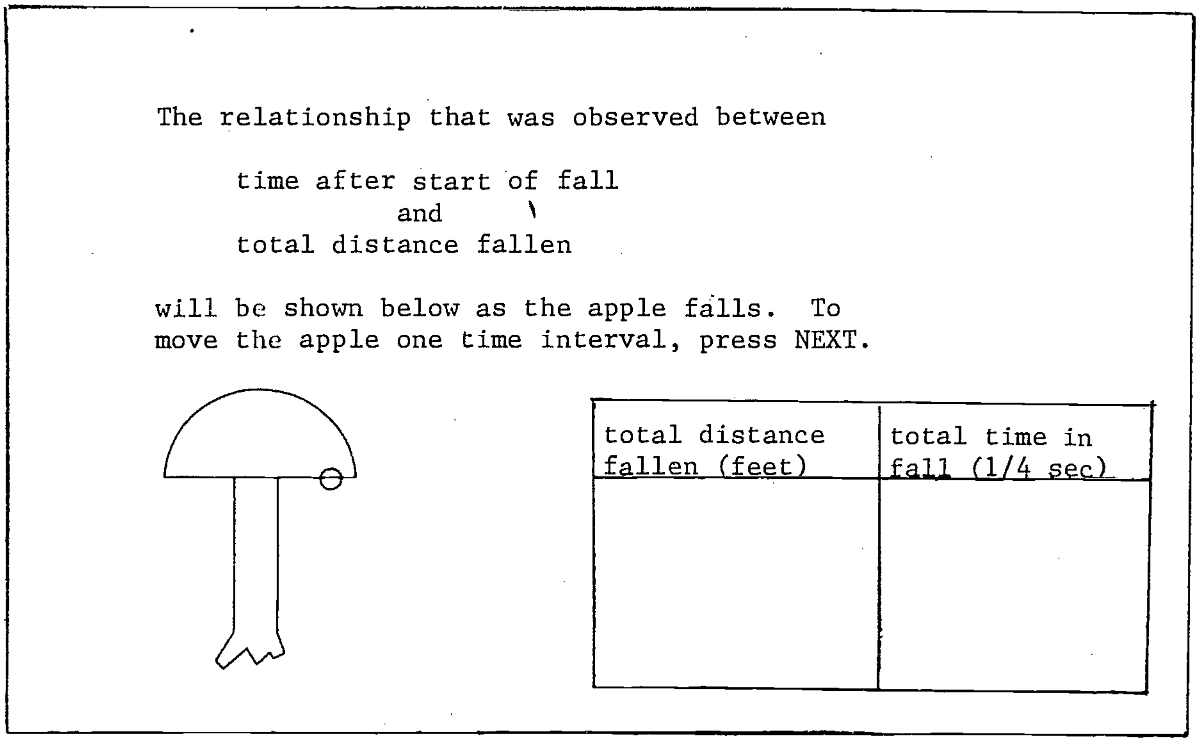
The revised unit "dropapp", figure 3.7, sets the initial values for the radius of the apple, the location of the apple, and the time which the apple has been falling. Before the apple is drawn at a new location, the apple at the old location must be erased. This is accomplished by unit "move". In unit "newapp" the variable "time" is incremented, and the location of the new apple is determined. The radius and the x-coordinate remain unchanged. The new y-coordinate is found by subtracting the amount of movement (time × time × 4) from the initial y-coordinate. (The "4" is merely a scaling factor. You may need a different value for your tree.) This is done in the tag of the -circle- command. Each time the apple is moved the same process must be repeated. Therefore several -do- statements are required in unit "dropapp". The essential features of the new physics unit, figure 3.8, are:

writing the text
drawing the tree
writing the table outline and headings
setting the radius and initial location for the apple
displaying the apples
displaying the data in the table

The data in this unit can be displayed at a location which is a function of some point of reference. An analogous structure was used in the units in figure 3.3 to display the table of data in figure 3.4. Use figures 3.3, 3.4 and 3.8 as well as the sample lesson as a guide in constructing your new units.

```
unit    dropapp  $$unit to drop the apple
calc    radius←4  $$initialize variables
        xcenter←178
        ycenter←224
        time←Ø
do      move  $$one -do- for each move of the apple
do      move
do      move
do      move
do      move
do      move
do      move
do      drawtree  $$patch up the tree
*
unit    move  $$this unit erases the old apple
at      xcenter-4,ycenter-4-time^2×4
erase   2
do      newapp
*
unit    newapp  $$this unit calculates new apple position
*                     and draws the new apple
calc    time←time+1
circle  radius,xcenter,ycenter-time×time×4
```

Figure 3.7 TUTOR code which drops the apple



The relationship that was observed between

time after start of fall
and
total distance fallen

will be shown below as the apple falls. To move the apple one time interval, press NEXT.

total distance fallen (feet)	total time in fall (1/4 sec)

Figure 3.8 Initial display for unit "newdata"

Psychology Units

For the psychology units, the number of incorrect components entered by the student will be tallied in a student variable named "pacerr". For an accurate count to be made, the defined variable must be set to Ø initially. It is convenient to do this in unit "pacintro". After each -wrong- command the error counter should be incremented by 1. After a correct response in unit "manson" an additional comment should be displayed: "You entered (display number of errors using -show-) incorrect components when responding to the psychology questions." To complete this exercise use the checklist provided in figure 3.9 as a reference when reviewing you lesson.

your checklist	Items to be checked in student mode
	Compare the sequence of units to the flow diagram, figure 3.6.
	Check the displays: Does your unit "newdata" resemble figure 3.8?
	Is the error message displayed correctly in unit "manson"?
	Check the calculations: Are the entries in the table for unit "newdata" correct"?
	Does "pacerr" count incorrect components entered by the student?

Figure 3.9 Checklist for Exercise C.

4. Conditional Operations

Conditional Branching

The use of variables can make a lesson quite flexible. Another tool for flexibility is conditional operations, that is, operations that are done only when a certain condition is met. Conditional commands in TUTOR are of the form

```
command expression,tagneg,tagØ,tag1,tag2,. . .
```

When a conditional command is encountered, PLATO evaluates the expression and rounds it to the nearest integer. If the integer is negative, tagneg is used. If the integer equals Ø, tagØ is used; if the integer equals 1, tag1 is used; and so forth. Below is a conditional -do- command.

```
do   v1,rivers,oceans,islands,x,plains,x
```

Upon encountering this command, PLATO determines the rounded value of v1. If this is a negative number, unit "rivers" will be done. If it equals Ø, unit "oceans" will be done; if it equals 1, unit "islands" will be done. If it is equal to 2 or is equal to or greater than 4, nothing will be done. The x's in the tag are not unit names, an x means "Do nothing for this value." Since x has this specialized meaning a unit cannot be named "x". The effect of the above -do- statement is summarized in figure 4.1.

```
do   v1,rivers,oceans,islands,x,plains,x
```

value of v1	unit which is attached	equivalent unconditional -do- statement
negative value	rivers	do rivers
Ø	oceans	do oceans
1	islands	do islands
1.2 (rounded to 1)	islands	do islands
2	no unit is attached	(no -do- statement)
2.5 (rounded to 3)	plains	do plains
3	plains	do plains
4 and greater than 4	no unit is attached	(no -do- statement)

Figure 4.1 Conditional -do- statement

Commas are used as separators in conditional commands. The last argument in the tag will be used if the rounded value of the expression is equal to or greater than the number of the last position in the tag. For example:

```
do   6÷v24-5,x,x,x,drawing
```

Unit "drawing" will be done if the expression, 6÷v24-5, is equal to or greater than 2.

Conditional Writing

Of the commands discussed so far, -next- and -do- can be used conditionally. There is a special form of the -write- command, -writec-, (pronounced: write see) for conditional writing. Here is an example.

```
writec  mistake,,You have made no errors.,You have made 1 er
        ror.,You have made several errors.
```

When "mistake" has a negative value, nothing is written. With the -writec- command nothing is written between the separators when no operation is desired. Another separator, ↕ , may be used instead of commas with a -writec- command. To type ↕ strike ACCESS (shifted ▭ , to the right of HELP), then comma. This allows the writing of sentences containing commas; e.g.

```
writec  verbtyp↕↕The verbs "faire", "aller", and "avoir" are irr
        egular.↕
        This is a regular verb.
```

Whichever symbol (, or ↕) immediately follows the expression will be the separator for the entire tag of the -writec- statement.

In unit "twain" below, different sentences are written on the panel depending on how many attempts the student has made to answer the question.

```
 1   unit    twain
 2   at      815
 3   write   Who wrote "The Prince and the Pauper"?
 4   arrow   1Ø13
 5   answer  Mark Twain
 6   write   That's right.
 7   answer  Samuel Clemens
 8   write   Correct.  His pen name was Mark Twain.
 9   no
1Ø   writec  ntries↕↕↕He also wrote "Life on the Mississippi".↕
11           He created the characters Aunt Polly and Injun Joe.↕
12           Hint:  His initials are M.T.
```

Figure 4.2 illustrates the unit in flow chart form. Lines 1 through 4 write the question and plot the arrow. Line 5 specifies a correct response. If the student's response matches the tag of line 5, line 6 will be executed. If it does not match, it is compared to the tag of the following answer-type command, in this case line 7. If they match, the response will be judged "ok" and the -write- command on line 8 will be executed. The -no- command on line 9 causes any unanticipated response to be judged "no". Commands below the -no- are executed if the student's response does not match any previous -answer- or -wrong- commands. If the student types anything other than "Mark Twain" or "Samuel Clemens", he will receive a "no" judgment, and the -writec- command will be executed. Which one of the sentences in the tag of the -writec- will be displayed depends on the value of the systems reserved word "ntries". If he has made one attempt, the sentence "He also wrote 'Life on the Mississippi'." will be written. If he has made two tries, he will see the sentence "He created the characters Aunt Polly and Injun Joe." If he has made three or more attempts, the sentence "Hint: His initials are M.T." will be displayed.

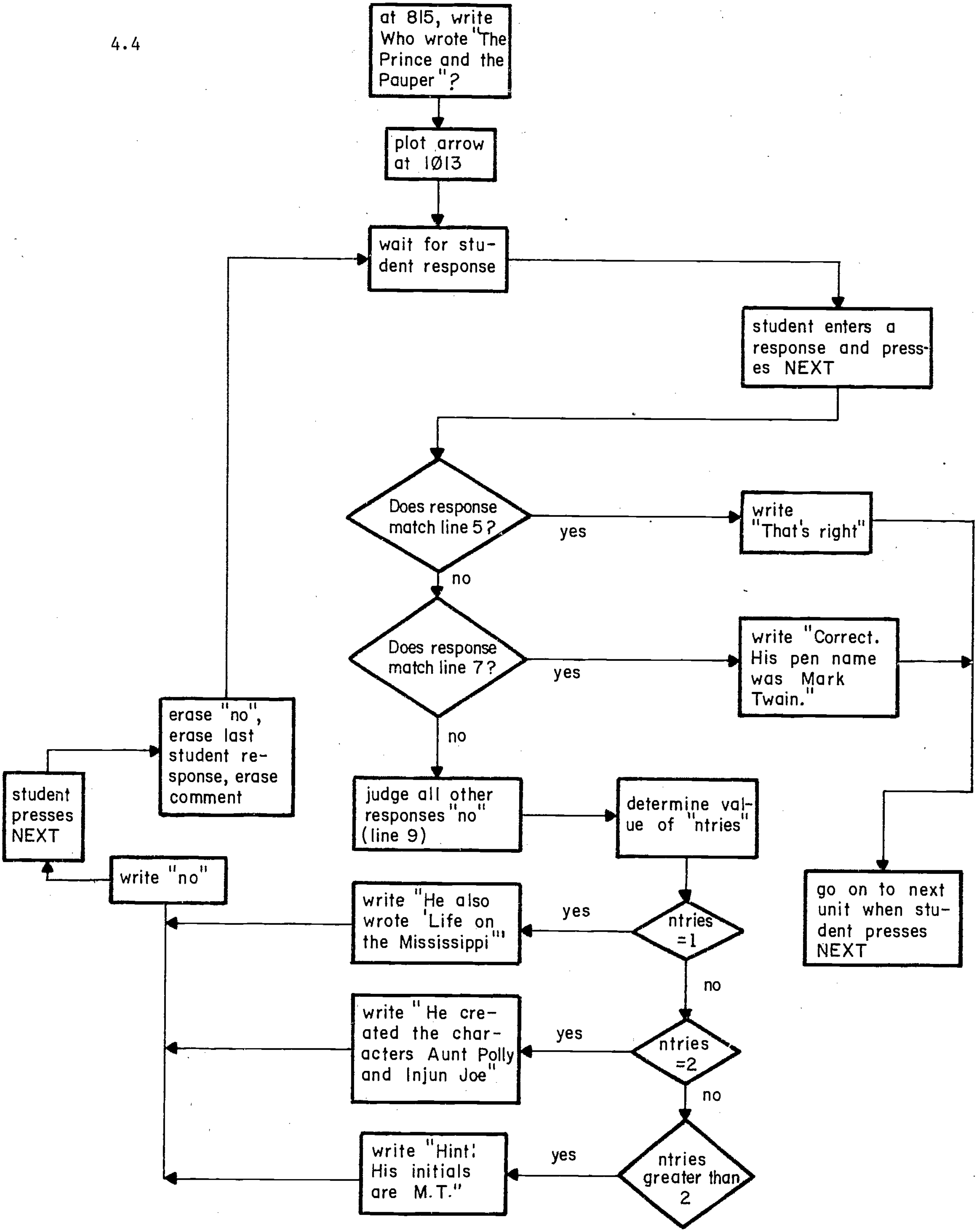
Exercise D - General Comments

The sequencing of the student through the units in exercise D is shown in the flow diagram, figure 4.3. A short review has been added to the psychology topic. Use sample lesson D and try these new units.

Psychology Units

In unit "pacintro" the student may select a short review. Depending on his response to the question "Would you like a brief review of how Harris analyzes transactions?" the student is branched to unit "numcomp" (for review) or unit "wifehus". This branching is done in the same way as the branching in unit "index".

The review branch consists of three new units, "numcomp" (for NUMber of COMPonents), "charcomp" (CHARacteristics of each COMPonent), and "interact". The units can be added in a new block called "pac review". Unit "numcomp" presents the question "Harris thought of each individual as having different



at 815, write Who wrote "The Prince and the Pauper"?
plot arrow at 1Ø13
wait for student response
student enters a response and presses NEXT
Does response match line 5?
yes
write "That's right"
no
Does response match line 7?
yes
write "Correct. His pen name was Mark Twain."
no
judge all other responses "no" (line 9)
determine value of "ntries"
go on to next unit when student presses NEXT
erase "no", erase last student response, erase comment
student presses NEXT
write "no"
ntries =1
yes
write "He also wrote 'Life on the Mississippi'"
no
ntries =2
yes
write "He created the characters Aunt Polly and Injun Joe"
no
ntries greater than 2
yes
write "Hint: His initials are M.T."

Figure 4.2 Flow diagram for unit "twain"

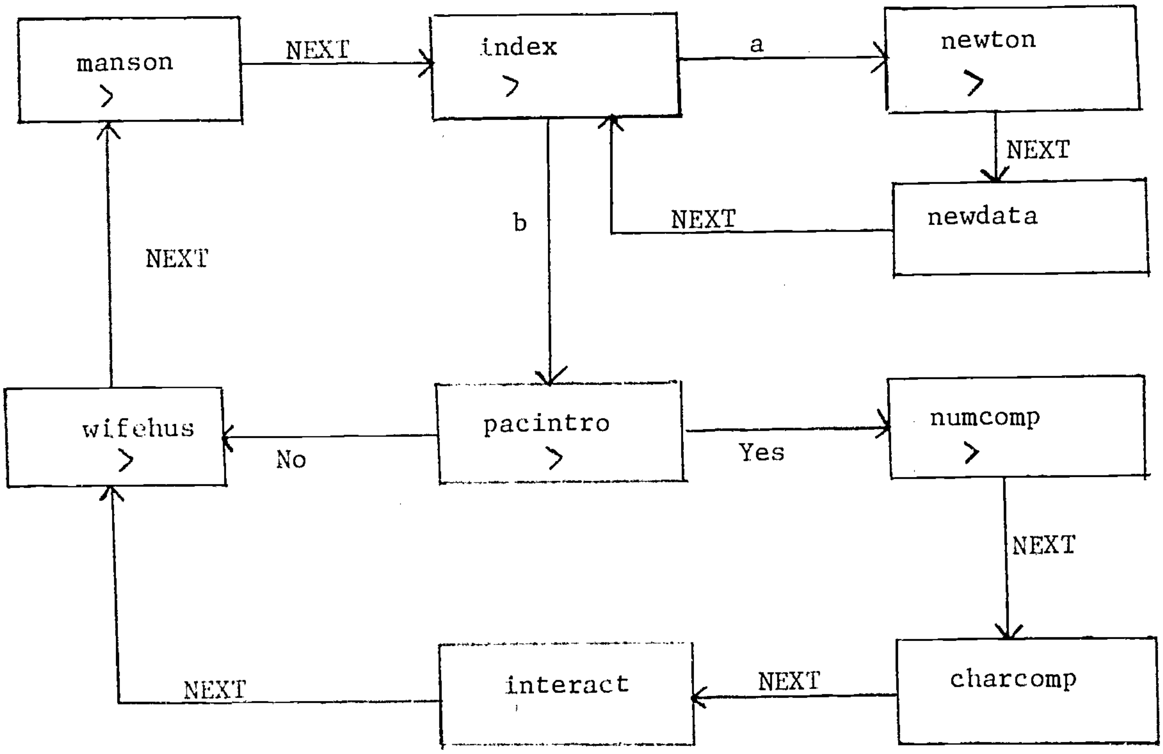


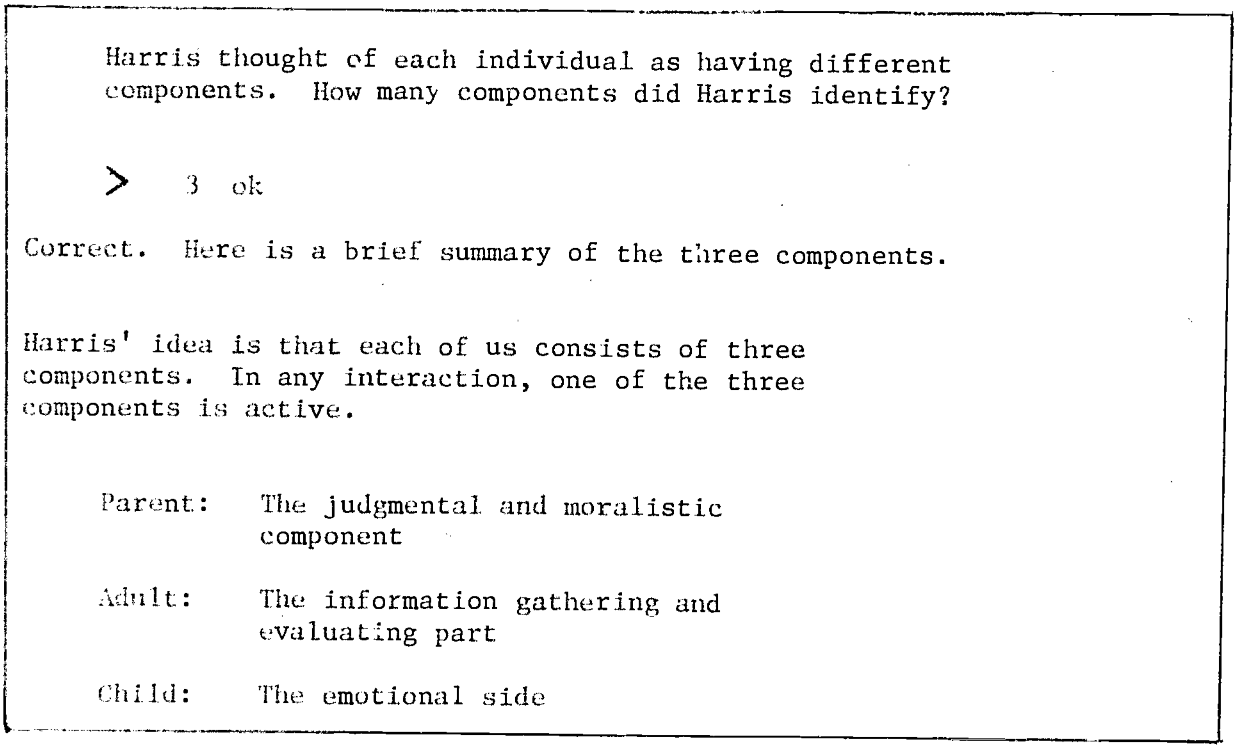
Figure 4.3 Flow diagram for Exercise D

components. How many components did Harris identify?" The judging and comments for this question are summarized in figure 4.4. Unexpected wrong responses are handled by a -no- command and a comment is written with a -writec- based on the number of attempts as given by "ntries". After a correct response, a comment is given to the student as presented in figure 4.5.

The last two review units consist of display material only. Unit "charcomp" should resemble figure 4.6, and unit "interact" should resemble figure 4.7. After unit "interact" the student should proceed to unit "wifehus" as indicated in the flow diagram.

Student Response	PLATO's Judgment	PLATO's Comment
3 three	ok	(refer to figure 4.5)
2, two 4, four	no	Close. Try again.
5, five 6, six 7, seven 8, eight 9, nine	no	A smaller number
all other responses	no	1 attempt: There are not many. 2 attempts: (no comment) 3 or more attempts: There are 3 components.

Figure 4.4 Response judging for unit "numcomp"



Harris thought of each individual as having different components. How many components did Harris identify?

> 3 ok

Correct. Here is a brief summary of the three components.

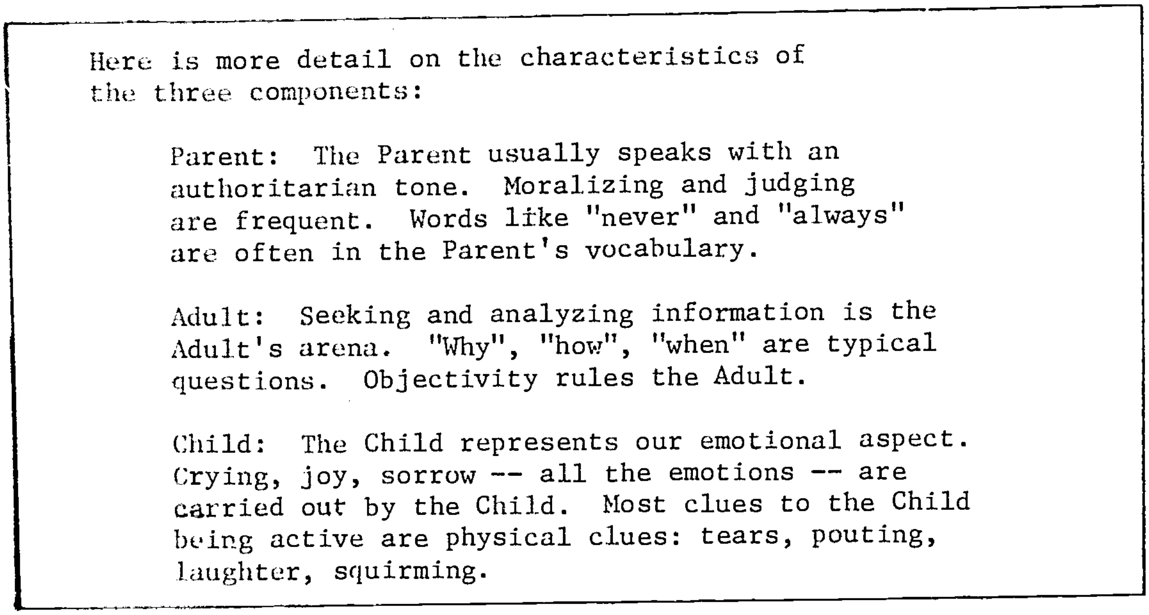
Harris' idea is that each of us consists of three components. In any interaction, one of the three components is active.

Parent: The judgmental and moralistic component

Adult: The information gathering and evaluating part

Child: The emotional side

Figure 4.5 Display after correct response in unit "numcomp"



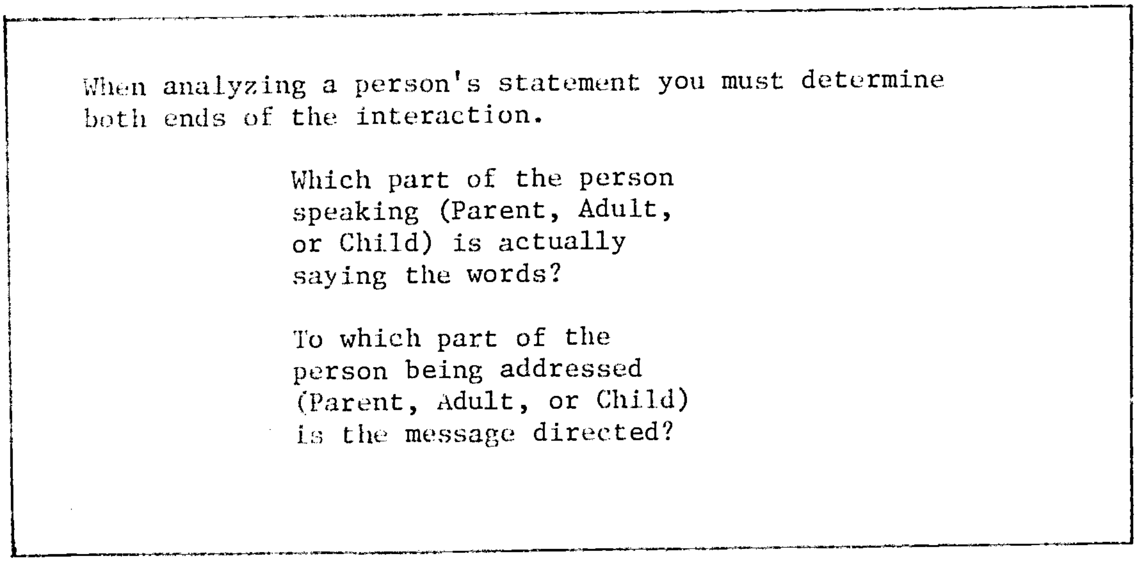
Here is more detail on the characteristics of the three components:

Parent: The Parent usually speaks with an authoritarian tone. Moralizing and judging are frequent. Words like "never" and "always" are often in the Parent's vocabulary.

Adult: Seeking and analyzing information is the Adult's arena. "Why", "how", "when" are typical questions. Objectivity rules the Adult.

Child: The Child represents our emotional aspect. Crying, joy, sorrow -- all the emotions -- are carried out by the Child. Most clues to the Child being active are physical clues: tears, pouting, laughter, squirming.

Figure 4.6 Display from unit "charcomp"



When analyzing a person's statement you must determine both ends of the interaction.

Which part of the person speaking (Parent, Adult, or Child) is actually saying the words?

To which part of the person being addressed (Parent, Adult, or Child) is the message directed?

Figure 4.7 Display from unit "interact"

Using the -no- Command

For each question in sample lesson D there is a comment given if the student enters an unanticipated response. The comments, which are summarized in figure 4.8, provide hints to the student. Since the same comment is used in units "wifehus" and "manson" you can write the comment in a unit and attach it in both places with -do- commands. In exercise C a tally of specific wrong answers for the psychology topic was kept. Since all incorrect answers are to be included in this tally, "pacerr" must be incremented after each -no- command. Follow the checklist in figure 4.9 when reviewing your lesson.

unit	PLATO's comment following -no-
index	Please enter a listed letter.
newton	Try a little bit of logic, please. Newton is under an apple tree. Apples are above him. One gets loose and falls...
pacintro	Please respond with "yes" or "no".
wifehus and manson	Please enter one of the components: Parent, Adult, or Child.
numcomp	(Use -writec- with "ntries") If "ntries" equals 1: There are not many. If "ntries" equals 2: (no comment) If "ntries" is 3 or greater: There are 3 components.

Figure 4.8 -no- comments for each question

your checklist	Items to be checked in student mode
	Compare the sequence of units to the flow diagram, figure 4.3.
	Do the new psychology units resemble figures 4.5, 4.6, and 4.7?
	Are the -no- comments correct, as in figure 4.8?
	Is "pacerr" incremented after -no- in psychology units?

Figure 4.9 Checklist for Exercise D

5. Branching the Student

Branching Via Function Keys

Branching commands in TUTOR are of two types: those which cause a branch when the command is encountered (-do-, -join-, -goto-, -jump-), and those which set up a branch path to be followed when the student presses a function key. The function keys are NEXT, BACK, HELP, LAB, DATA, TERM, shift-NEXT, shift-BACK, shift-HELP, shift-LAB, and shift-DATA. The shifted function keys are abbreviated NEXT1, BACK1, etc. The commands which activate the function keys have the same names as the keys, i.e. -next-, -next1-, -help-, -help1-, etc.

When a student presses one of the function keys he is branched to a new unit if the author has so specified. For example:

```
unit    verbs
next    adverbs
back    nouns
at      252Ø
write   Press NEXT to go on, BACK
        to review
.
.
.
```

If the student presses NEXT he goes on to unit "adverbs". Pressing BACK takes him to unit "nouns". Pressing any of the other function keys has no effect because there are no corresponding commands in this unit. As we saw in Chapter 2, however, NEXT is always active; and if there is no -next- command then the unit physically following the present one is the next unit.

Main Units

Each time the student enters a unit as a result of pressing a function key (e.g. HELP, NEXT) or because of a -jump- command, the student has entered a new main unit. When each main unit is entered the plasma panel display is erased. Units which are attached to a main unit by -do-, -join-, or -goto- commands (-join- and -goto- will be discussed in Chapters 7 and 11) are called attached units. In the example code below, the main unit is "newton", and the attached unit "drawtree" has been inserted into unit "newton" by the -do- command.

```
unit     newton
at       2Ø6
write    One day Newton was sitting under a tree...
do       drawtree
.
.        more TUTOR code
.
unit     drawtree
*        a tree is displayed with -draw- command
```

The -jump- Command

An immediate branch to a new main unit can be created with the -jump- command. The tag of the -jump- command is the name of the unit to "jump" to. When a -jump- command is encountered, the student is sent immediately to the unit named in the tag, without pressing a key. The -jump- command may be used conditionally. In the sequence of units in figure 5.1, the student is immediately branched, with the -jump- command, to one of three different units, depending on his choice.

If the student's response matches the tag of one of the -answer- commands, he will be sent immediately to the corresponding unit. A response other than "a", "b", or "c" will be judged "no" and the message "Please choose a, b, or c" will be displayed.

Help Sequences: Establishing a Base Unit

Branches by the HELP, LAB, DATA, HELP1, LAB1, and DATA1 keys differ from branches by NEXT, NEXT1, BACK, and BACK1, in that the first group (collectively referred to as help-type keys) initiate help sequences. In a help sequence, the unit the student was in when he pressed the help-type key is "remembered" and the student is returned to that unit when the help sequence is ended. The unit in which the help-type key was pressed is called the base unit. The end of a help sequence is marked by an -end help- statement. Pressing NEXT in a unit containing an -end- command, or pressing BACK or BACK1 in any unit in a help sequence, returns the student to the base unit. The units in figure 5.2 illustrate a help sequence.

```
unit    index
at      52Ø
write   Choose a topic:
at      81Ø
write   a.  Trigonometry
        b.  Graphing
        c.  Algebra
arrow   1515
specs   bumpshift
answer  a
jump    trig
answer  b
jump    graph
answer  c
jump    algebra
no
write   Please choose a, b, or c.
*
unit    trig
.
.       more TUTOR code
.
unit    graph
.
.       more TUTOR code
.
unit    algebra
.
.
.       more TUTOR code
.
```

Figure 5.1 An index using -jump-

```
unit    jef
next    madison
help    hint
at      81Ø
write   Who was the third president of the United States?
arrow   111Ø
at      151Ø
write   If you need a hint, press HELP.
specs   bumpshift,okextra
answer  jefferson
*
unit    hint
next    hint2
at      9Ø3
write   He authored the Declaration of Independence.
at      15Ø5
write   Press NEXT for more help;
        Press shift-BACK to return to the question.
**
unit    hint2
at      9Ø5
write   He campaigned vigorously for religious freedom and c
        ivil
        rights.
at      15Ø5
write   Press NEXT to return to the question.
end     help
**
unit    madison
at      81Ø
write   Which president followed Jefferson?
arrow   111Ø
specs   bumpshift,okextra
answer  madison
```

Figure 5.2 Code for four units on presidents

If the student presses the HELP key while he is in unit "jef" he is branched to the unit named in the tag of the -help- command, unit "hint". If in unit "hint" he presses NEXT, he will go on to unit "hint2". From "hint2" pressing NEXT returns him to the base unit, "jef", because of the -end help- statement in "hint2". A flow chart of these units (figure 5.3) illustrates what unit the student will be in when he presses various keys.

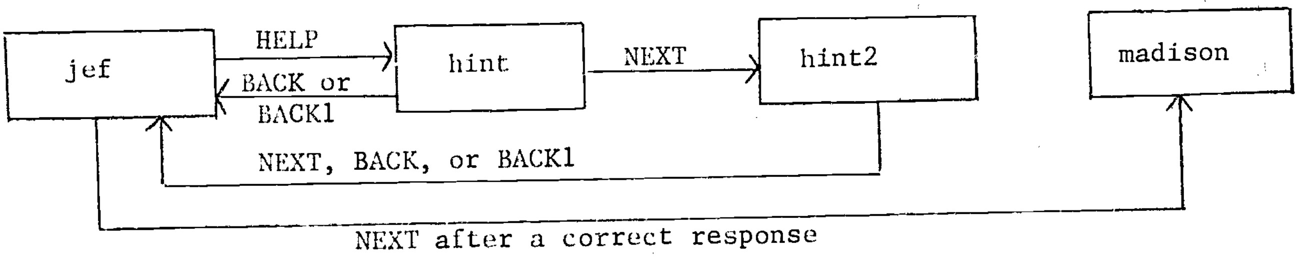


Figure 5.3 Flow chart for units on presidents

All branching commands may be used conditionally. The unit in figure 5.4, unit "metric", uses one of two different help sequences, depending on how many errors the student has made. (Assume the variable "wrongs" has been defined, and has been used to count the number of errors made by the student in an earlier part of the lesson.) In this example only students who have made 2 or more errors will have a help sequence available. Those who have 2 errors will be sent to unit "table" if they press HELP; those with 3 or more errors will be sent to unit "explain". The x's in the tag of the -help- command specify that no HELP branch will be made for students who have fewer than 2 mistakes. The message "HELP is available" is written only if "wrongs" is 2 or greater. Students who have made no mistakes, or only one, will not see the help message. With conditional commands, the expression is evaluated and the branch path for possible later use is established when the command is encountered, not when the branching key is pressed.

When the student enters unit "metric" his variable "wrongs" is evaluated. If he presses HELP he will proceed to unit "table" if the value in "wrongs" was equal to 2, or to unit "explain" if the value was greater than 2. Notice the -end- commands in each help unit. Pressing NEXT from either of the units causes a return to the base unit, "metric". BACK and BACK1 also return the student to the base unit.

```
unit    metric
help    wrongs,x,x,x,table,explain  $$"wrongs" was tallied earlier
at      8Ø5
writec  wrongs,,,,HELP is available
at      12Ø5
write   What is the equivalent in British pints of 4 liters?
arrow   14Ø5
.
.
.       more TUTOR code
.
.
unit    explain
at      9Ø5
write   To convert liters to British pints, multiply by 1.75
        98Ø4
end     help
*
unit    table
.
.
.       some TUTOR code to display a table of
.       conversion factors
.
.
end     help
```

Figure 5.4 Help sequence used conditionally

Units can be thought of as main units or attached units, main units being those reached by a keypress or a -jump- command and attached units being those reached by -do-, -join-, or -goto-. The term "base unit" is applicable only when the student is in a help sequence. The base unit is the unit in which he pressed the help-type key. When the help sequence is terminated and the student returns to the unit from which he entered the help sequence, he no longer has a base unit.

In figure 5.5, all main units are represented by rectangles. Units attached to the main units by -do- (-join- or -goto-) commands are circles. When a student is in a main unit with a * in the upper left corner, the base unit pointer is set because these main units were reached by pressing the DATA key. The base unit is the unit from which the DATA key was pressed.

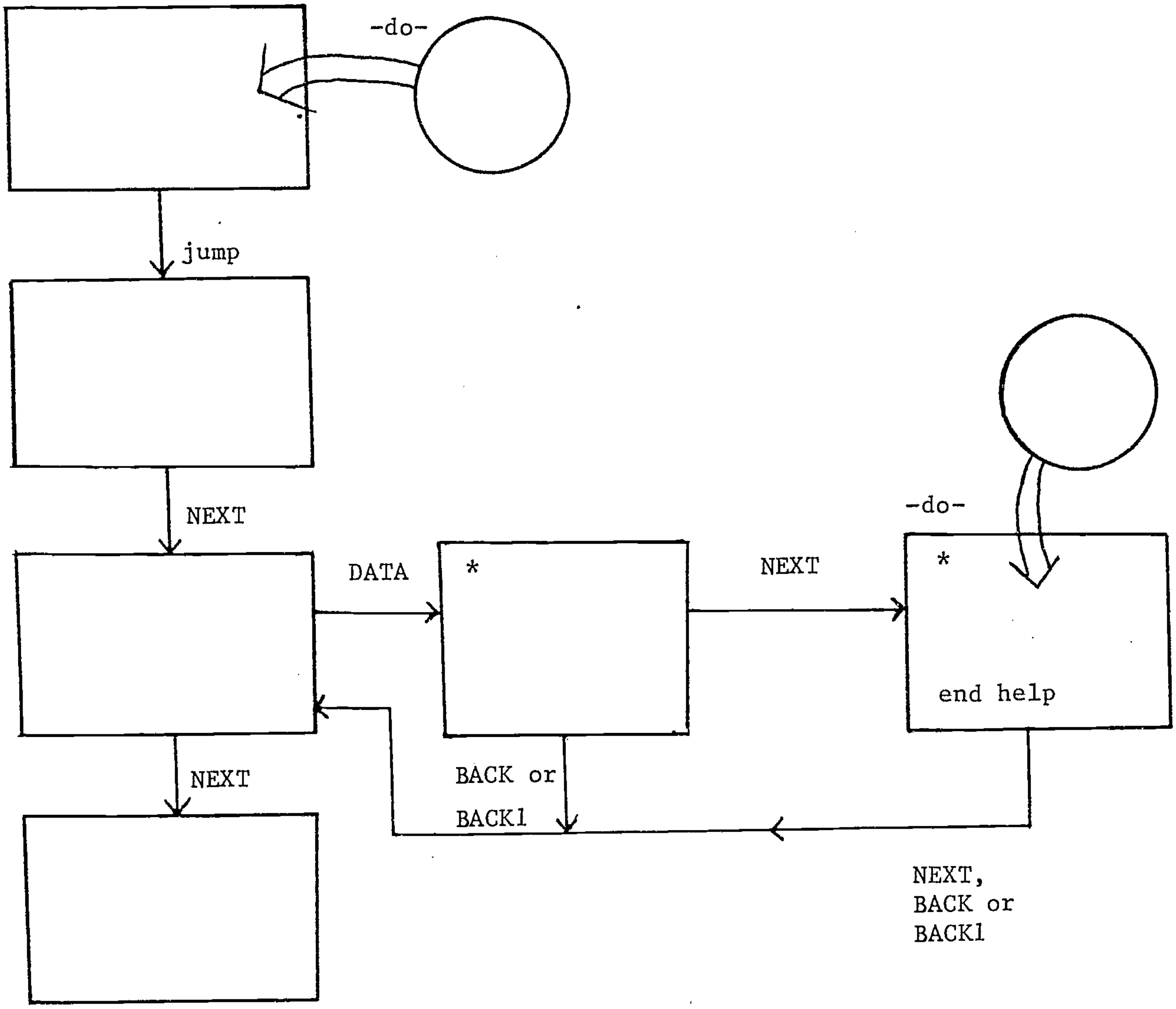


Figure 5.5 Main, base, and attached units

Exercise E - General Comments

The review units for the psychology topic are now used in three different ways, as indicated in the flow diagram, figure 5.6. These units still function as a review based on the student's response to the question in unit "pacintro". The units also serve as a help sequence from units "wifehus" and "manson". The third use of this review material is as a compulsory review if the student has made 3 or more errors when responding to the transactional analysis questions. The sequencing of the student through the review units for the optional branch from unit "pacintro" and the forced branch from unit "manson" is controlled through the use of student variables.

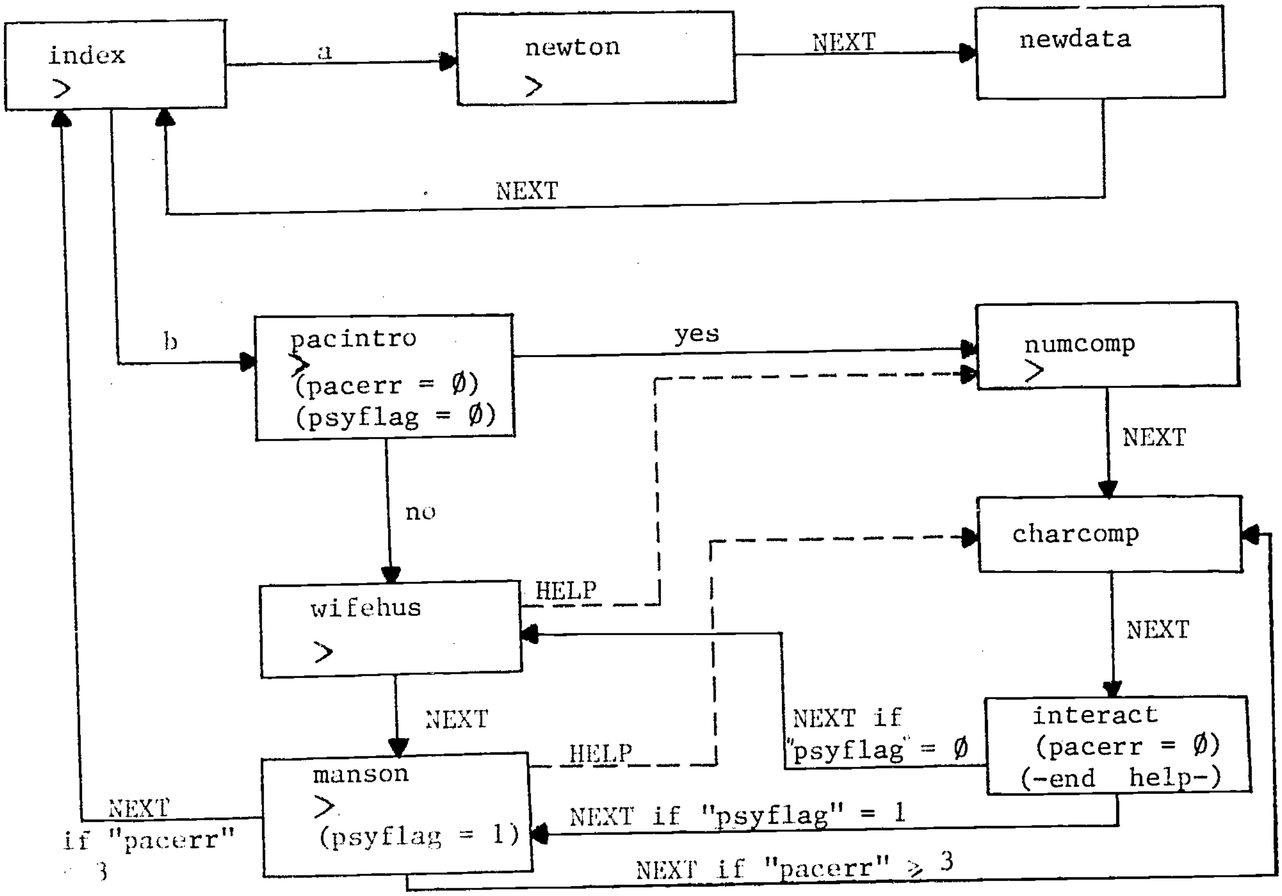
Use of the -help- Command

From unit "wifehus" if the student presses HELP, he should proceed to unit "numcomp", then unit "charcomp", then "interact", and from there he should be returned to his base unit ("wifehus"). If the student presses HELP in unit "manson", he should be sent to unit "charcomp", then unit "interact". After "interact" he should return to unit "manson", his base unit. Unit "interact" is the last unit in both help sequences, so it will need an -end help- statement as the last statement in the unit. Appropriate -help- commands should be inserted in your lesson. It is helpful for the student to know when the HELP key is active; therefore the message "HELP is available" is displayed in units "wifehus" and "manson".

Using Student Variables as "flags"

Student variables have been used extensively in the tags of display commands, in calculations, and in tallying errors. In the psychology units, student variables will be used to determine the student's path through the lesson material. A new defined variable named "psyflag" and the variable "pacerr" are used for this purpose.

The variable "pacerr" is set to ∅ in unit "pacintro". After the scudent has answered both transactional analysis questions, he is conditionally branched,



After HELP is requested in either units "wifehus" or "manson", pressing BACK or BACK1 will end the help sequence. If still in effect, the help sequence will end with unit "interact" because of the -end- command and the student will return to his base unit. This branching is not included on the flow diagram.

Figure 5.6 Flow diagram for Exercise E

based on the current value of "pacerr", either to the index or to a short review starting with unit "charcomp". Pressing NEXT should take the student to unit "index" if "pacerr" is Ø, 1, or 2; but if "pacerr" is 3 or greater, pressing NEXT should take the student to unit "charcomp". The conditional -next- statement

```
        next    pacerr-3,index,charcomp
```

should be placed after the -answer- command, not at the beginning of the unit. When the -next- command is encountered the "NEXT unit" is established based on the condition of the "pacerr" expression. If this -next- command appeared at the beginning of unit "manson", the "NEXT unit" would be established at the time unit "manson" was entered, and the effect of any changes to "pacerr" in this unit would not be noticed.

If the student makes 3 or more errors in the interaction questions he will be forced to study the review units. The counter ("pacerr") should be set to Ø during the review. Therefore, when the student returns to the question in unit "manson", the conditional -next- statement will branch him according to the number of errors made this time in unit "manson". If "pacerr" were not zeroed in the review units, the student could never exit from the psychology topic.

The variable "psyflag" is set to Ø in unit "pacintro". The value remains Ø until unit "manson" at which time it is set to 1. In unit "interact", the last review unit, "psyflag" is used to determine what unit will be the "NEXT unit". If the student entered the review units from "pacintro" then "psyflag" will be Ø and unit "wifehus" will follow unit "interact". If he entered the review units from "manson" then "psyflag" will be 1 and unit "manson" will follow unit "interact". This branching can be done by a conditional -next- command in unit "interact" using "psyflag" as the index expression. Use the flow diagram as a reference for the conditional branching with student variables.

The comment after a correct response in unit "manson" has been changed. Instead of being shown the number of errors he made, the student is shown a comment based on the value of "pacerr". The comments are shown in figure 5.7. To complete exercise E, try all the items on the checklist, figure 5.8.

Value of "pacerr"	Comment by PLATO
∅	Very good analyses of these transactions!
1, 2	This completes the psychology topic.
3 and greater	Let's review some of Harris' ideas and then return to the transactions.

Figure 5.7 Comments after correct response in unit "manson"

your checklist	Items to be checked in student mode
	Try the HELP key in units "wifehus" and "manson".
	When unit "interact" is the last unit of a help sequence, is the student returned to the base unit?
	Does the "HELP is available" message appear in units "wifehus" and "manson"?
	In unit "manson" is the student appropriately branched as a function of "pacerr"?
	In unit "interact" is the student appropriately branched as a function of "psyflag"?
	Are the comments in unit "manson" displayed as in figure 5.7?

Figure 5.8 Checklist for Exercise E

6. The Judging Process

Regular and Judging Commands

If a -write- command is placed after -answer- or -wrong- commands it is done only if the student's response matches the tag of the -answer- or -wrong- command. TUTOR commands are classified into two categories, regular commands and judging commands. Of the commands in Chapters 1 to 5, the -specs-, -answer-, -wrong-, and -no- are judging commands; the rest are regular commands. At any particular time, PLATO is executing either regular commands or judging commands, but not both. When executing regular commands, PLATO is said to be in "regular state"; when executing judging commands, it is in "judging state".

```
1    unit     capital
2    at       81Ø
3    write    What is the capital of New York?
4    arrow    11Ø8
5    answer   Albany
6    write    That's right.
7    wrong    New York <City>
8    write    It's the largest city, but not the capital.
9    unit     geog
```

PLATO begins this unit and executes the regular commands -at- and -write- (Lines 2 and 3). It then encounters the -arrow- command, notes its location in the unit, and plots an arrow on the panel. Still in regular state, it looks for any more regular commands between the -arrow- command and the first judging command. If any are found they are executed, but in this unit the command following the -arrow- command is -answer-, a judging command. PLATO is in regular state and does not execute a judging command, so PLATO stops and waits for the student to enter a response and press NEXT. When the student presses NEXT, PLATO is switched into judging state. Now PLATO goes back to the -arrow- command, the location of which was noted before, and looks for the first judging command after the -arrow-. The first judging command, -answer-, is on line 5. PLATO executes this command by comparing the student response to the tag. If they don't match, PLATO continues down the unit, still in judging state. The following command is -write- (line 6). Since -write- is a regular command, it is not executed in judging state. The following command,

-wrong-, is a judging command and it is executed. If no match is found, PLATO looks at the following command, -write-, and skips it because it is a regular command. The -unit- command (line 9) stops the search for any more judging commands in this unit "capital". Since no match was found, PLATO writes "no" after the student's response. When the student presses NEXT again, his response and the "no" are erased.

Now PLATO is again waiting for the student to press NEXT to reinitiate judging. When he enters a new response and presses NEXT, PLATO executes the -answer- command again. If the student's response does not match the tag, the -write- command on line 6 is skipped (because PLATO is again in judging state) and the next judging command, -wrong-, is executed. If the student's response matches the tag of the -wrong- command, PLATO is switched back to regular state. This time the -write- command on line 8, a regular command, is executed. Matching of a -wrong- tag also causes "no" to be written after the student's response. The following command is -unit-. However, PLATO will not leave unit "capital" until the arrow has been satisfied with an "ok" judgment. When the student presses NEXT or ERASE after an incorrect response, his response is erased and PLATO returns to the -arrow- and waits again.

When the student enters another response and presses NEXT again, the whole process is repeated. Judging commands are executed and regular commands are skipped. If the student's response matches the tag of the -answer- command, PLATO switches to regular state for any regular commands between the matched -answer- command and the next judging command. The -write- statement on line 6 is the only regular command found. It is executed and "ok" is written after the student's response. The arrow has been satisfied so now pressing NEXT will take the student to unit "geog".

The process just described can be summarized as follows. PLATO begins a main unit in regular state. All regular commands are executed until an -arrow- command is encountered. The place of the -arrow- command in the unit is noted and PLATO continues to execute regular commands until a judging command is encountered. Then PLATO waits for a student response. When the student presses NEXT, PLATO switches to judging state. The first judging command after the -arrow- is executed. If there is no match, subsequent judging commands are executed, until a -unit- or -endarrow- command (described later in this Chapter) is encountered. If none of the judging commands are matched the response is judged "no" and PLATO returns to the arrow. If an answer-type or wrong-type command is matched PLATO is switched to regular state. Regular commands after the matched command and before the next judging command or -unit- or -endarrow- command are executed. An appropriate judgment is given, "ok" or "no", and pressing NEXT takes the student to the next unit (if "ok") or back to the arrow (if "no"). The flow diagram, figure 6.1, summarizes the operation of a TUTOR unit.

Judging Numerical Responses

Besides the -answer- and -wrong- commands, TUTOR has commands for judging numerical responses, -ansv- and -wrongv-. Here is an example of their use.

```
unit    math
at      81Ø
write   4 × 8 =
arrow   817
ansv    32
write   Correct.
wrongv  12
write   Mulitply; don't add.
```

If the student types 12, he will see the comment "Multiply; don't add." If he types 32, he will see the comment "Correct".

A tolerance of an absolute or a percent deviation can be specified with the -ansv- and -wrongv- commands.

```
 1  unit    math
 2  at      81Ø
 3  write   4 × 8 =
 4  arrow   817
 5  ansv    32
 6  write   Correct.
 7  wrongv  32,2
 8  write   You're off a little.
 9  wrongv  12
1Ø  write   Multiply; don't add.
```

The second argument in the -wrongv- tag on line 7 is the tolerance allowed. If the student types 3Ø, 31, 33, or 34 he will see the comment "You're off a little." The response "32" will match the -ansv- statement (line 5) and not the -wrongv- statement (line 7), since the -ansv- precedes the -wrongv-.

Operation of a TUTOR Unit -- interaction of regular and judging commands

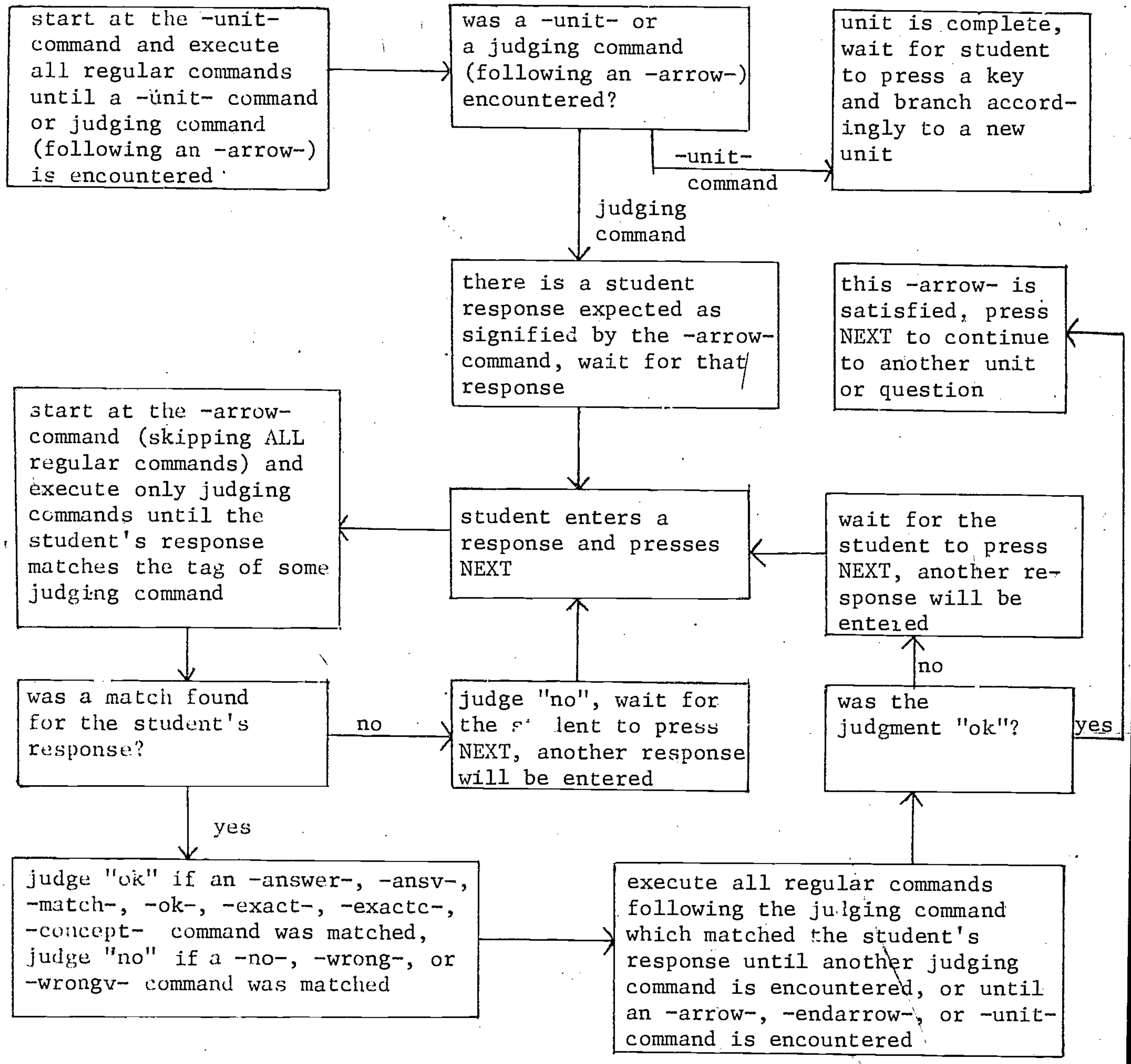

start at the -unit- command and execute all regular commands until a -unit- command or judging command (following an -arrow-) is encountered
was a -unit- or a judging command (following an -arrow-) encountered?
-unit- command
unit is complete, wait for student to press a key and branch accordingly to a new unit
judging command
there is a student response expected as signified by the -arrow- command, wait for that response
this -arrow- is satisfied, press NEXT to continue to another unit or question
start at the -arrow- command (skipping ALL regular commands) and execute only judging commands until the student's response matches the tag of some judging command
student enters a response and presses NEXT
wait for the student to press NEXT, another response will be entered
no
was a match found for the student's response?
no
judge "no", wait for the s' lent to press NEXT, another response will be entered
was the judgment "ok"?
yes
yes
judge "ok" if an -answer-, -ansv-, -match-, -ok-, -exact-, -exactc-, -concept- command was matched, judge "no" if a -no-, -wrong-, or -wrongv- command was matched
execute all regular commands following the ju lging command which matched the student's response until another judging command is encountered, or until an -arrow-, -endarrow-, or -unit- command is encountered

Figure 6.1 Flow diagram of a TUTOR unit

Here is an example of -ansv- and -wrongv- with a percent deviation tolerance of 1Ø%. Any percent deviation may be specified.

```
unit    distance
at      81Ø
write   What is the circumference of the earth at the
        equator, in miles?
arrow   121Ø
ansv    25ØØØ,1Ø%
write   That's quite a distance!
wrongv  8ØØØ,1Ø%
write   That's the diameter.
no
write   Try again.
```

Any response from 225ØØ to 275ØØ will be judged "ok". A response from 72ØØ to 88ØØ will match the tag of the -wrongv- and the student will see "That's the diameter." Any other response will be judged "no", and the comment "Try again." will appear.

It is important to know that PLATO evaluates the student's response as an expression, and compares the result with the tag of the -ansv-. In unit "distance" if the student types an expression equal to 25ØØØ, it will be judged "ok". If the student used scientific notation, and typed "2.5 × $1Ø^4$" or "25 × $1Ø^3$" he would be judged "ok". In other words, PLATO evaluates the student's arithmetic expression and then compares it to the tag of the -ansv- or -wrongv- command.

However, sometimes you may not want to allow this. In the example unit "math", the student is asked to perform the calculation, and we would not want to allow the response "4 × 8" to the question "4 × 8 = ". You can use

```
specs   noops
```

to specify that the student's response may not contain math operators. The systems reserved word "opcnt" contains the number of operators in the student's response. Below unit "math" is modified to use this -specs- option and systems reserved word.

```
unit    math
at      81Ø
write   4 × 8
arrow   817
specs   noops
ansv    32
write   Correct.
no
writec  opcnt,,,Do not use an operator in your answer.
        Give a constant.
```

If the student uses any operators in his response, -specs noops- will cause a "no" judgment and "opcnt" will be 1 or greater. The comment "Do not use . . ." will be written.

Storing Numerical Responses

You may want to store a numerical response in a student variable. The stored value can then be used for branching or other purposes. Unit "choice" below uses the stored student response to branch to the unit the student wants to study. The -store- command (a judging command) stores a number or the value of an expression typed by the student in the variable named in the tag. Unlike -answer- and -wrong-, -store- does not cause a switch to regular state after a number has been stored; judging state continues. There is a switch to regular state with a "no" judgment only if the student's response is not "storeable"; that is if it is an illegal arithmetic expression or is something other than a number or an arithmetic expression.

```
unit    choice
at      5Ø3
write   Press the number of the topic you want to study.
at      81Ø
write   1.  The Archeology of Troy
        2.  The Trojan War
        3.  Odysseus' Return from Troy
arrow   12Ø5
store   part
no
**the command above judges all responses "no"
jump    part,x,x,arch,troywar,oreturn,x
at      15Ø5
write   You must choose 1, 2, or 3.  Press NEXT and enter
        another number.
```

The number the student types is stored in the variable "part". If the student enters 1, 2, or 3, he does not see the "no" since he is immediately branched by the -jump- command, line 12. If a number other than 1, 2, or 3 is entered the x's in the tag of the -jump- will cause PLATO to "fall through" to the following -write- command. Use of the -no- command and the conditional -jump- allows the desired branching, but prevents the erasure and rewriting of the panel after an inappropriate response.

Multiple Arrows in a Unit

Suppose you want to ask the student two related questions, such as "Who was the third president of the U.S.?" and "Who was vice-president under him?" If these questions are in two different units, the first one will be erased when the student enters the second unit. It would be nice to have the two questions appear on the same page, as is done in unit "pres".

```
 1   unit      pres
 2   at        8Ø5
 3   write     Who was the third president of the U.S.?
 4   arrow     1ØØ3
 5   answer    <T,Thomas> Jefferson
 6   write     You're  right.
 7   wrong     <J,John> Adams
 8   write     He was the second president.
 9   endarrow
1Ø   at        16Ø5
11   write     Who was vice-president under him?
12   arrow     18Ø3
13   answer    <A,Aaron> Burr
14   write     Right again!
15   wrong     <B,Ben,Benjamin> Franklin
16   write     He was an ambassador, but never vice-president.
17   *
18   unit      senators
```

The -endarrow- command on line 9 delimits the portion of the unit pertinent to the first -arrow-. The initial question is written at 8Ø5. PLATO encounters an -arrow- command and notes its location in the unit. It then continues to execute any regular commands that occur before the first judging command. There are none, so PLATO returns to the -arrow- and waits for the student to type a response and press NEXT. When he does, PLATO begins the search for judging commands. If there is no match to the -answer- tag (line 5) or the -wrong- tag, PLATO keeps looking for another judging command.

But encountering the -endarrow- means, in effect, "Stop looking for any more judging commands." So the response is judged "no" and PLATO returns to the arrow to wait for another student response. In judging state, the -endarrow- command has acted like a new -unit- command, stopping the search for more judging commands.

When the first arrow has been satisfied the -endarrow- command will cause the arrow plotted on the screen at 1ØØ3 to be erased, but the question, the student's response, and comment "You're right" will remain on the panel. PLATO then proceeds to process the rest of the unit.

Exercise F - General Comments

The sequencing of the psychology units is the same as in exercise E, and one unit has been added to the physics topic. The psychology units are therefore omitted from the flow diagram, figure 6.2.

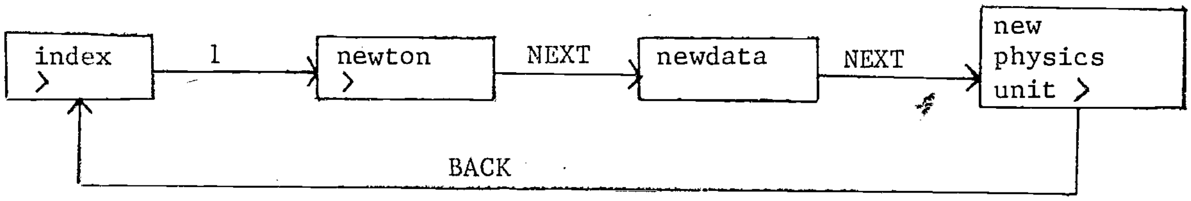


Figure 6.2 Partial flow diagram for Exercise F

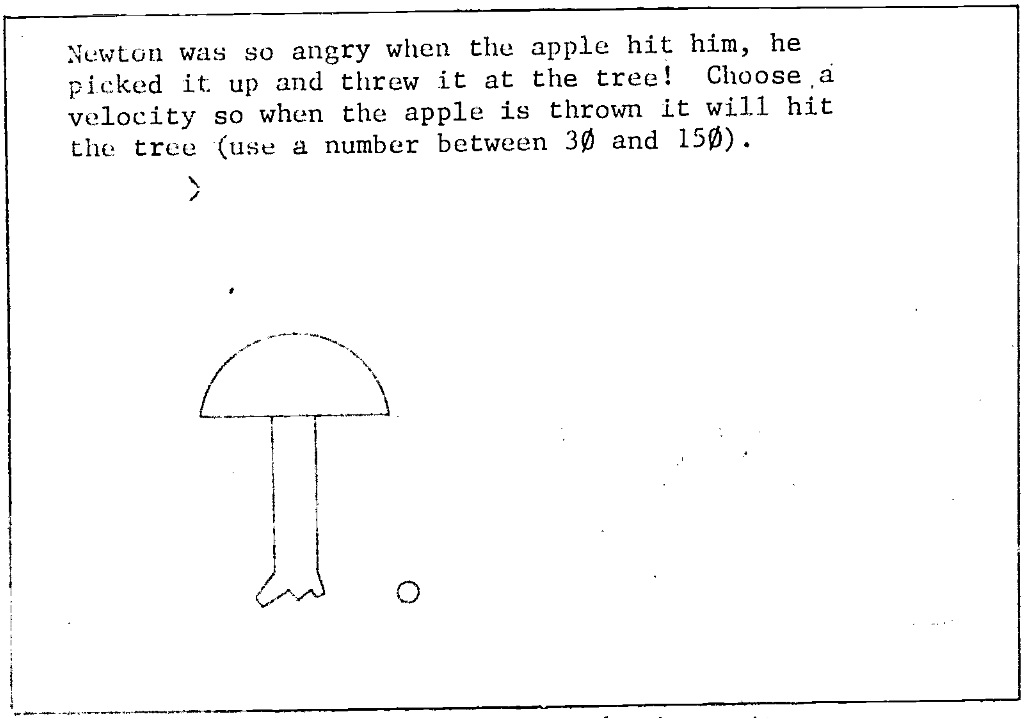
The Index

The choices from the index are now designated by numbers instead of letters. Numbers are used so that a conditional -jump- command can be executed based on the student's answer as stored by a -store- command. The judging for unit "index" is similar to that of unit "choice", p. 6.6.

Physics Units

The new physics unit has the student select a velocity for Newton to use when throwing the apple back at the tree. For this unit, directions to the student are needed, the familiar apple tree has to be displayed, and Newton needs an apple sitting on the ground. Your display for this unit should resemble figure 6.3.

The student responses can be grouped into four categories: a velocity large enough so that the apple hits the tree; a velocity which is not large enough; a velocity outside certain limits (the limits are the approximate range of a person's pitching capability); a velocity which is not a number or a legal expression. If the student enters a response which is not "storeable" by a -store- command, PLATO automatically judges "no" and there is a switch to regular state. A -write- command can be placed below the -store- command to give an appropriate comment if the student's response is not a legal expression or a number. If the velocity specified by the student is outside the limits, a -no- command followed by a -write- command can direct the student to velocities within the specified limits.



Newton was so angry when the apple hit him, he
picked it up and threw it at the tree! Choose a
velocity so when the apple is thrown it will hit
the tree (use a number between 3Ø and 15Ø).

>

Figure 6.3 Display for new physics unit

If the velocity entered is within the specified limits, whether large enough to actually hit the tree or not, the following should be done:

```
erase the apple on the ground
display the apple in the air
write a comment (see figure 6.4)
erase the apple in the air
display the apple on the ground
```

If the tossed apple did not hit the tree, the student can press NEXT to erase the comment, and then enter a new value. If the apple did hit the tree, the student has the option of either pressing NEXT so another toss can be made, or branching to the index by pressing BACK.

After a "no" judgment PLATO automatically erases the last comment displayed when the student presses NEXT to enter another response. This feature is used in the sample les on by using a -wrongv 9Ø,6Ø- statement for judging all entered velocities between 3Ø and 15Ø. However it is not appropriate for the student to see the "no" for each response within these limits, so a -specs nookno- statement is used.

Did the apple hit the tree?	Comment
yes	WOW! You hit the tree for Newton. Press NEXT to try another toss, or BACK to return to the index.
no	A larger velocity is needed to hit the tree.

Figure 6.4 Comments displayed if velocity is within specified limits

Since all responses in this unit are judged "no", an exit is needed. A conditional -back- statement is used for this purpose. Pressing the BACK key only branches the student to another unit; no help sequence is initiated. The conditional expression for the -back- and the -writec- statements (used for the comments in figure 6.4) can be determined by considering the distance between the apple and the tree's branches, and the total distance the apple will reach for a given initial velocity specified by the student. The relationship between total distance and initial velocity is:

$$\text{total distance} = \frac{\text{initial velocity}^2}{64}$$

For example, if the distance between the apple and the branches is 256 dots (4 × 64 dots), an initial velocity of 128 (2 × 64) is required for the apple to hit the tree. To compare the student's initial velocity with the distance of 256 dots in this example, a statement

```
back    256-(initvel²÷64),x,index
```

will activate the BACK key if the defined variable "initvel" is large enough to have the apple hit the tree.

Psychology Units

In units "wifehus" and "manson", a second question is added. An -endarrow- command is used to delimit the judging commands for the first -arrow- and the presentation and judging for the second -arrow-. After the second -arrow- another -endarrow- is used so there can be a break between the judging for the second -arrow- and the summary comments and drawing presented after the second -arrow-. A -pause- command is used after each -endarrow- so the student can continue at his own rate after an "ok" judgment.

The general format for units "wifehus" and "manson" is:

```
display interaction and initial question
response judging for initial question
-endarrow-
-pause-
display second question
```

```
response judging for second question
-endarrow-
-pause-
erase help message, questions, and responses
display initial comment
draw interaction pointer
-pause-
display second comment
proceed to appropriate unit when NEXT is pressed
```

Summaries for the response judging and the comments after the second question is "ok" are presented in figures 6.5 and 6.6. As with the initial questions for these units, the defined variable "pacerr" should be incremented after each incorrect response. The conditional -next- statement should now follow the last -endarrow- in unit "manson". This position of -next- is important because the student errors in this unit are considered when PLATO "decides" if review is needed by using the current value of "pacerr". A checklist for this exercise is provided in figure 6.7.

unit	question	response	judgment	comment
wifehus	1	parent	ok	none
	1	adult	no	The wife's response (sarcastic tone) contains more than information about the tie.
	1	child	no	If it were the child, it would be more self-centered.
	1	all other responses	no	Please enter one of the components: Parent, Adult, or Child.
	2	child	ok	none
	2	adult	no	Not with the sarcasm. She is trying to do more than give information.
	2	parent	no	She is not appealing to a moralistic or judgmental aspect of him.
	2	all other responses	no	Please enter one of the components; Parent, Adult, or Child.
manson	1	adult	ok	none
	1	child	no	There was no emotional reeling in the father's response.
	1	parent	no	A Parent's response probably would have scolded the son for being afraid or not trying hard enough.
	1	all other responses	no	Please enter one of the components; Parent, Adult, or Child
	2	child	ok	none
	2	parent	no	No appeal was made to morals or right or wrong.
	2	adult	no	The father's statement would include a stronger attempt to reassure his son.
	2	all other responses	no	Please enter one of the components: Parent, Adult, or Child

Figure 6.3 Response judging for units "wifehus" and "manson"

unit		comment
wifehus		Here is the wife's part of this interaction.
	-pause-	Harris calls this type of interaction a crossed interaction. It leads to difficulties because the individual is responding with a different component than was addressed. Tempers soon flare because of the failure to communicate.
manson		This example shows what Harris calls a complementary transaction. This type of transaction can continue indefinitely because the two components are communicating.
	-pause-	(the -writec- statement based on "pacerr"; refer to figure 5.7)

Figure 6.6 Comments after second question in units "wifehus" and "manson"

your checklist	Items to be checked in student mode
	Compare the sequence of units to the flow diagram, figure 6.2.
	Does the new judging for unit "index" branch the student correctly?
	Does the display for the new physics unit resemble figure 6.3?
	Are the comments for the new physics unit displayed according to figure 6.4?
	Is the student allowed to exit from the new physics unit with the BACK key only after the apple toss hits the tree branches?
	Are the response judging specifications for the new psychology questions in accordance with figure 6.5?
	Are the summary comments in figure 6.6 displayed with appropriate pauses?
	Is "pacerr" incremented after each "no" judgment for the new psychology questions?
	Does the conditional -next- in unit "manson" follow the last -endarrow-?

Figure 6.7 Checklist for Exercise F

7. Random Numbers

Sampling with Replacement

PLATO can generate random numbers. The statement

```
randu   v1Ø,9
```

generates an integer from 1 to 9 (the number given in the second argument of the tag) and stores it in v1Ø (the location named in the first argument of the tag). The generation of random numbers by the -randu- command is "sampling with replacement". That is, each number within the specified range has the same chance of being selected as any other number in the specified range.

The unit "multiply", figure 7.1, generates two numbers and calculates their product. The student is shown the numbers and asked to multiply them. His response is compared to PLATO's result and judged accordingly. After doing five items correctly on the first try he is jumped to unit "done".

In unit "setup" the variable "right" is set to Ø. This variable will serve as a counter: 1 will be added to its value every time the student answers correctly on the first attempt. It must have the value Ø when the student starts the drill so that it will contain the number of questions correct on the first attempt. The -next- command in unit "multiply", line 7, specifies that after this unit, unit "multiply" is to be done again. The student will cycle through this unit until he has answered 5 items correctly on the first attempt. The -jump- command on line 8 is encountered each time unit "multiply" is begun. It is a conditional command, so PIATO looks at the value in "right". The first time it contains Ø, and since an x appears in the "Ø" position of the tag, there is a "fall through" to the following command, the -randu- on line 9. A number from 1 to 7 is generated and stored in "first". Then (line 1Ø) another number from 1 to 8 is generated and stored in "second".

The problem is displayed using embedded -show- commands (lines 11 and 12) and the arrow is plotted. When the student enters his response and presses NEXT, it is compared (line 14) with the tag of the -ansv- statement. If they match and if "ntries" equals 1, unit "counter" is executed (line 15). The value in "right" is incremented by 1, making the value now equal to 1. If "ntries" is not equal to 1, "right" is not incremented. When the student presses NEXT he starts through unit "multiply" again. When the -jump- command on line 8 is encountered, the value of "right" is checked. If it is less than 5 there is a "fall through". Two new numbers are generated and stored in "first" and "second" (lines 9 and 1∅). These numbers might be the same as the ones previously chosen. When the student enters a new response it is again compared with the tag of the -ansv-. If they match and if "ntries" equals 1, PLATO adds 1 to the value in "right". If they do not match or if "ntries" is greater than 1, the old value remains in "right". The student keeps going through this unit until the value in "right" equals 5. When "right" contains the value 5, encountering the -jump- command on line 8 causes an immediate branch to unit "done".

The -randu- command may also have a one-argument tag. The statement

```
randu   v1∅
```

causes a fraction between ∅.∅ and 1.∅ to be randomly generated and stored in v1∅.

```
 1   unit    setup
 2   **"first", "second", and "right" have been defined
 3   zero    right
 4   jump    multiply
 5   *
 6   unit    multiply
 7   next    multiply
 8   jump    right-5,x,done  $$falls through if "right" is less than 5
 9   randu   first,7
1∅   randu   second,8
11   at      41∅
12   write   ⊲s,first⊳ × ⊲s,second⊳ =
13   arrow   where+3
14   ansv    first×second
15   do      ntries,x,x,counter,x
16   wrongv  first:second
17   write   You are dividing.
18   *
19   unit    done
2∅   at      615
21   write   You have finished the exercise.
22   *
23   unit    counter
24   calc    right⇐right+1
```

Figure 7.1 Random number units

Sampling without Replacement

The -randu- command gives sampling with replacement; that is, the same number may be generated more than once. It is sometimes desirable to sample without replacement; i.e., to generate numbers randomly, but not to pick the same number twice. The -setperm- and -randp- commands can do this. The statement

```
setperm 6
```

sets up the list from which the numbers will be taken. The number in the tag is the largest number in the list. In other words, the above statement sets up a number field from 1 to 6. The statement

```
randp   v16
```

picks a number from the field set up by a previous -setperm- command, places that number in v16, and deletes that number from the setperm list.

The -setperm- and -randp- commands can be used to design a drill that presents items in random order. The units in figure 7.2 are a foreign language vocabulary drill. A new command is introduced in this drill, -join-. The -join- command works like the -do- command, except that it is executed in judging state as well as in regular state. It will be used to attach units containing -answer- commands to the main unit.

In unit "set" a field of 5 numbers is established (line 3) from which to pick at random. The tag of the -setperm- command here is a constant; it may be a variable. Then the student is jumped to unit "lang". The statement

```
next    lang
```

sets up the same kind of looping structure that was used earlier in the discussion of the -randu- command. The -randp- command (line 8) picks a number from 1 to 5, places it in "prob", and deletes it from the setperm list. After all numbers have been picked once, the -randp- command will place Ø in "prob". The -jump- command on line 9 serves to branch the student out of unit "lang" when the field has been exhausted; that is when the drill is complete. If "prob" contains a number greater than Ø, the student will continue in unit "lang".

```
 1   unit    set
 2   **variable "prob" has been defined in i.e.u.
 3   setperm 5  $$ set up list: 1, 2, 3, 4, 5
 4   jump    lang
 5   **
 6   unit    lang
 7   next    lang
 8   randp   prob $$ pick a number from setperm list, delete it from list
 9   jump    prob,x,done,x  $$ when Ø returned, drill is done
1Ø   at      81Ø
11   write   Translate into French:
12   at      1Ø15
13   writec  prob,,,a horse,a house,a book,a hat,a street
14   arrow   1213
15   join    prob,x,x,a1,a2,a3,a4,a5
16   *
17   unit    a1
18   answer  un cheval
19   unit    a2
2Ø   answer  une maison
21   unit    a3
22   answer  un livre
23   unit    a4
24   answer  un chapeau
25   unit    a5
26   answer  une rue
27   *
28   unit    done  $$ done with drill
29   *       more TUTOR code
```

Figure 7.2 Random drill; no review of "missed" items

Instructions are written on the panel (lines 1Ø and 11). The -writec- command on line 13 writes one of the items on the panel depending on the value in "prob". When the student signals PLATO to begin judging, the -join- command attaches the appropriate -answer- statement. The attached unit depends again on the value in "prob", and since the value in "prob" has not changed, the unit joined will be the one appropriate to the word displayed by the -writec- command on line 13.

Automatic Review of Items "missed" in a Drill

Frequently in a drill of this type you may want to present again to the student the items he missed. The language drill can be altered to give review of the items missed by including two new commands, -modperm- and -remove-, and the systems reserved word "ntries". The actions of -modperm- and -remove-can best be explained by more discussion of -setperm-. When you write the statement

```
setperm 5
```

PLATO sets up two fields like this:

setperm list	1	2	3	4	5
modperm list	1	2	3	4	5

Then when a -randp- command is encountered, one of those numbers is chosen, and taken out of the setperm list but not out of the modperm list. So if 3 is picked, the fields look like this:

setperm list	1	2		4	5
modperm list	1	2	3	4	5

The randomly chosen number is always removed from the setperm list by the -randp- command. You can selectively remove numbers from the modperm list with the -remove- command. The form of the command is

```
remove   v16
```

where the tag is the variable in which the random number is stored by the -randp- command. When a -modperm- command is executed the contents of the modperm list are placed in the setperm list.

You can remove from the modperm list the numbers of all the items in a drill for which the student met a certain criterion, such as answering correctly on the first or the second attempt. Then, after the student has seen all problems once, you can use the -modperm- command to present again to the student all the items he missed. Figure 7.3 presents the French translation drill, modified to present the items in random order and then present again the ones which the studrnt did not answer correctly on the first try.

The instructions and the problem are displayed and the student's response is compared to the tag of the appropriate -answer- command. When judging is ended, line 17 is executed. If "ntries" equals 1 and the student's response matches the tag of the -answer- command, unit "removal" is attached (lines 19

```
unit    set
**variable "prob" has been defined in i.e.u.
setperm 5  $$ establish modperm and setperm lists
jump    lang
**
unit    lang
next    lang
randp   prob
do      prob,x,review,x  $$ attach "review" if setperm list is empty
jump    prob,x,done,x
at      81Ø
write   Translate into French
at      1Ø15
writec  prob,,,a horse,a house,a book,a hat,a street
arrow   1213
join    prob,x,x,a1,a2,a3,a4,a5
do      ntries,x,x,removal,x  $$ remove from modperm list if ntries = 1
*
unit    removal
remove  prob  $$ delete number from modperm list
*
unit    review
modperm        $$ place modperm list into setperm list
randp   prob
*
unit    a1
answer  un cheval
unit    a2
answer  une maison
unit    a3
answer  un livre
unit    a4
answer  un chapeau
unit    a5
answer  une rue
*
unit    done
*       more TUTOR code
```

Figure 7.3 Random drill with review of "missed" items

and 2Ø). That is, the number associated with this item is removed from the modperm list.

If "prob" is equal to Ø (line 8), unit "review" is attached, line 9, and the modperm list is placed in the setperm list by the -modperm- command, line 23. A number is chosen from the new setperm list with the -randp-, line 24. If that number is Ø, meaning that all the numbers from the new setperm list have been picked, the student proceeds to unit "done" (line 1Ø), because he has completed all the items correctly on the first attempt. If "prob" does not equal Ø, he continues in unit "lang" as before. This structure will be repeated until all numbers have been removed from the modperm list; that is, until the student has done all the items right on tne first try.

A setperm list may not be larger than 12Ø items. However TUTOR has commands which make it possible to work with more than 12Ø items or to sample from more than one list at a time. These commands are described in lesson "aids" (see Chapter 13).

8. Additional Judging Capabilities

The -specs- Command as a Marker

It is often convenient to have the -specs- command immediately following the -arrow- command. However, the -specs- command can be placed anywhere after the -arrow- command and there can be more than one -specs- command for each arrow.

```
1    arrow    1422
2    specs    okextra
3    answer   Homo Sapiens
4    write    very good
5    specs    okextra,bumpshift
6    answer   (woman,man,human)
7    no
8    write    woman and man
```

For the first anticipated -answer-, capitalization is important, therefore only the "okextra" tag (line 2) is included. If the student's response does not match the -answer- statement in line 3, PLATO will continue to search for judging commands. The -specs- command on line 5 is the next judging command. This second -specs- command clears the old "specifications" set by the first -specs- command, and establishes new "specifications" for the judging commands following line 5 in the above example.

The -specs- command also acts as a marker which is returned to after PLATO has made a judgment. All regular commands following the last -specs- command encountered for the current -arrow- are executed after each judgment for that -arrow-. This processing of regular commands stops when a judging command is encountered.

```
1    arrow    152Ø
2    at       1ØØ6
3    write    Who was the first President?
4    specs    okspell,okextra
5    at       22Ø6
6    write    His home was Mt. Vernon, Virginia.
7    answer   Washington
8    write    He was quite a soldier.
9    wrong    Jefferson
1Ø   calc     errors←errors+1
```

After the arrow is displayed on line 15, space 2∅, the question "Who was the first President?" is displayed, then PLATO encounters a judging command (-specs-) and waits for the student to enter a response. After a response is entered and NEXT is pressed, PLATO returns to the -arrow- and searches for judging commands. The -specs-, -answer-, and -wrong- commands are the only judging commands, so they are processed in that order until a match for the response is found. After a judgment by PLATO, the regular commands following the -specs- command (lines 5 and 6) are executed until a judging command is encountered, line 7. The possible student responses, judgments, comments, and calculations after a judgment are summarized in figure 8.1.

Response	Judgment	Comments and Calculations
Washington	ok	He was quite a soldier. His home was Mt. Vernon, Virginia.
Jefferson	no	calc errors←errors+1 His home was Mt. Vernon, Virginia.
all other responses	no	His home was Mt. Vernon, Virginia.

Figure 8.1 Summary of writing after -specs-

A -specs- command with a blank tag can function as a marker to be returned to after a judgment for that -arrow- has been made. Such an example is presented later in this chapter, figure 8.5. A -specs- command with a blank tag also clears previous "specifications" established by -specs- commands since the current -arrow- was encountered.

The -match- Command

A response judging command which is more specialized than the -answer- command is the -match- command. The -match- command looks for words in the student's response that are in the tag of the -match- command, and automatically sets a variable based on which argument of the -match- tag contained one of the words in the student's response. If a word in the student's response is in the tag of the -match- command, PLATO judges the response as "ok"; otherwise a "no" judgment is given.

```
        match   v3,horse,(pig,swine),cow
```

If the student entered the response "horse", the student variable v3 would be set to Ø and the response would be judged "ok". With a response of either "pig" or "swine" (synonyms are in parentheses separated by commas), v3 would be set to 1 and an "ok" judgment would be made. If the student entered the word "cow", the response is judged "ok" and v3 would equal 2. If some other response was entered, for example "sheep", that response would be judged "no" because the word "sheep" does not appear in the tag of the -match- command, and the variable v3 would be set to -1.

If the student's response is found in the tag of the -match- command, the variable is set to the value corresponding to the relative location of the "matched" word and an "ok" judgment is made. The relative locations of the tag of the -match- command are numbered starting with zero and proceeding through the positive integers. If no match is made, a value of -1 is always placed in the variable (first argument) named in the tag of the -match- command and a "no" judgment follows. The -match- command always terminates judging state because it either finds a student response (and judges "ok") or no response is found (and it judges "no") and some value is always stored in the variable named in the tag of -match-. The automatic checks and mark-up for word order, spelling, and extra words which are done with -answer- are not part of the -match- command.

The value of the variable which is set as a result of executing a -match- command can be used for branching the student. If an index is displayed, as in figure 8.2, and the student chooses either topic a, b, or c, he should be branched to the appropriate units. If another choice is made, such as f, a "no" judgment should follow with the comment: "Please choose either a, b, or c." The response judging for the index of figure 8.2 is shown below.

```
        match   choose,(a,A),(b,B),(c,C)
        jump    choose,x,cemnom,molwt,balequat
        write   Please choose either a, b, or c.
```

If no response is found, then "choose" is set to -1, a "no" judgment is made, the "x" or fall through argument of the -jump- is done, and the -write- statement is displayed. If a response is found, "choose" is set to either ∅, 1, or 2 (depending on the response) and the -jump- command takes the student to a new unit. If a response is found, the -write- command is never executed. This process is summarized in figure 8.3. If a -specs bumpshift- statement was inserted before the -match- command, then the synonyms (a,A), etc. would not be needed.

Please choose one of the topics with the corresponding letter

a. Chemical Nomenclature

b. Molecular Weight Determination

c. Balancing Equations

Figure 8.2 Typical index of topics

student response	judgment	value of "choose"	jump to unit
a or A	ok	∅	cemnom
b or B	ok	1	molwt
c or C	ok	2	balequat
anything else	no	-1	"fall through" to -write-

Figure 8.3 Judgment by -match- and value of "choose"

In unit "verbs", figure 8.4, the student enters a subject pronoun. Which pronoun he entered is determined with a -match- command, and the corresponding verb form is displayed to the right of the pronoun. A new systems reserved word, "jcount", is introduced. It contains the number of characters in the student's response. It is used here to position the verb.

The student will cycle through the unit as many times as he wants. When he enters a response the capitals are removed because of the "bumpshift" tag of -specs-. The automatic writing of "ok" or "no" is prevented by the "nookno" tag of the -specs- command. His response is searched for any of the strings listed in the tag of the -match- command. If the string "i" is found, the variable "pronoun" is set to Ø; if "he", "she" or "it" is found, "pronoun" is set to 1, if "you", "we" or "they" is found, "pronoun" is set to 2. If the response matches none of the listed strings, "pronoun" is set to -1. The position of the writing is determined (it will be at 1615, plus the length of the word the student entered, plus 3). Then the appropriate verb is written; or if "pronoun" equals -1, the student is told that his response is not a subject pronoun.

```
unit     verbs
next     verbs
lab      gerunds
inhibit  erase
at       1615
erase    4Ø
at       1Ø1Ø
write    Enter a subject pronoun and PLATO will conjugate the
         verb "to be" with it.  Press LAB when you are ready
         to go on.
arrow    1615
specs    bumpshift,nookno
match    pronoun,i,(he,she,it)(you,we,they)
*        The variable "pronoun" must be previously defined.
at       1615+jcount+3
writec   pronoun,This is not a subject pronoun.,am,is,are
```

Figure 8.4 Using -match- and "jcount"

The "judging copy" of a Response

When the student enters a response at an arrow, a copy of the response, called the "judging copy", is made. It is the judging copy that TUTOR compares to the tags of -answer-, -ansv-, -wrong-, etc. commands. For example, when a -specs bumpshift- is encountered, the capitals are removed from the judging copy of the response, but the capitals remain on the student's display panel. When the student's response matches an entry in the tag of a -match- command, that entry is removed from the judging copy of the response and replaced with blanks.

In unit "convert", figure 8.5, the student is asked to convert inches to centimeters. A new command is introduced in this unit, the -judge- command. It is a <u>regular</u> command and can be used to modify PLATO's automatic judgment. There are several possible tags for the -judge- command and it can be used in a conditional form, as in line 8. Some of the tags of the -judge- command are summarized in figure 8.6.

```
 1   unit     convert
 2   *        "inches", "units", and "number" are defined in i.e.u.
 3   calc     inches←1Ø  $$ for a drill, this may be a -randu-
 4   at       61Ø
 5   write    convert ⋖s,inches⋗ inches to centimeters.
 6   arrow    1Ø15
 7   specs
 8   judge    units,no,x,no
 9   writec   units↕please enter your units↕↕
1Ø            you are converting from inches to cm
11   at       161Ø
12   writec   (units+1)×judged↕very good↕↕
13            2.54×⋖s,inches⋗ ≠ ⋖s,number⋗↕↕
14   *
15   match    units,(cm,centimeter,centimeters),(in,inches,
16            inch)
17   judge    continue
18   store    number
19   ansv     2.54×inches
```

Figure 8.5 Example of -match- and -judge-

tag	brief summary
continue	switch to processing judging commands
ok	judge the ~~response~~ "ok"
no	judge the response "no" (unanticipated)
wrong	judge the response "nc" (anticipated)
x	do not alter PLATO judgment

Figure 8.6 Some tags for -judge-

A new systems reserved word, "judged", is also introduced in unit "convert". It can have the following values:

-1 after an "ok" judgment
∅ after an anticipated "no" judgment
1 after an unanticipated "no" judgment

An anticipated "no" judgment means a response that was judged "no" as a result of matching a -wrong-, -wrongv-, or encountering a -judge wrong- statement. An unanticipated "no" judgment is a result of matching the -no- command, not matching any judging commands, or encountering a -judge no- statement.

In unit "convert" the -match- command identifies and removes the units, either cm. or in., from the judging copy of the student's response. After the -match- command, line 15 in figure 8.5, the numerical part of the student's response must still be evaluated. However after a -match- command, PLATO has made a judgment (either "ok" or "no") and is now processing regular commands. Before the numerical part of the response can be evaluated with the -ansv- command, PLATO must be instructed to cease processing regular commands and to start processing judging commands again. The -judge continue- statement, line 17, does just that. Thus starting with line 18, PLATO is again processing judging commands, using the judging copy of the student's response altered by the -match- command. The -store- command places the numerical expression in "number" and the -ansv- command evaluates the numerical part of the student's response.

After the -ansv- command, PLATO has again made a judgment (either "ok" or "no") because the -ansv- is the last judging command for the current -arrow-. The -specs- command on line 7 of figure 8.5 is used only as a marker to be returned to after PLATO has processed all judging commands for the current -arrow-. Upon returning to the -specs- command, PLATO is processing regular commands so lines 8 through 13 are executed after each judgment for this -arrow-. When a judging command is again encountered, line 15, PLATO ceases the execution of regular commands.

The -judge- command on line 8 of figure 8.5 will judge any student response "no" if the units are not stated as cm. If the numerical part of the student's response was correct (as evaluated by the -ansv-) but the units were wrong, the conditional -judge- command would set the PLATO judgment to "no". Thus, the -judge- command can be used to alter a previous judgment of PLATO.

Some typical student responses are analyzed in figure 8.7. Unit "convert" contains several new (and complex) ideas. The logic of this example is intended to be only a sample of TUTOR code. Some major programming concepts of this unit are summarized in figure 8.8.

	student response	value of "units",line 15	judgment, lines 19&8	value of "judged",line 12	comment, line 9-13
1.	25.4 cm	Ø	ok	-1	very good
2.	25.4	-1	no	1	please enter...
3.	2Ø cm	Ø	no	1	2.54 × 1Ø≠...
4.	2.54 in	1	no	1	you are...

classification of student response

1. correct number and unit entered
2. no unit entered with any number
3. incorrect number entered with correct unit
4. incorrect unit entered with any number

Figure 8.7 Typical responses for unit "convert"

1. -specs- used as only a marker
2. conditional -judge- command
3. use of reserved word "judged"
4. return to judging state with -judge continue-
5. altering a previous judgment of PLATO

Figure 8.8 New concepts in unit "convert"

Removing Numbers from the Judging Copy

The -storen- command removes arithmetic expressions, one at a time, from a student's response and places the expression in the variable named in the tag of the -storen- command. The removed expression is replaced with blanks in the judging copy of the response. If a student was asked "How tall are you?", the numbers could be acquired by:

```
1   arrow   151Ø
2   storen  feet
3   write   Please state your height as a number.
4   storen  inches
5   write   Please enter both the feet and inches.
6   specs   okextra
7   answer  (ft,feet) (in,inches)
8   write   thank you
9   no
1Ø  write   Use the form: 5 feet 8 inches.
```

The first -storen-, line 2, places the feet value in the defined variable "feet" and removes that number from the judging copy. The second -storen- removes the inches value after placing it in "inches". Then the remaining words can be interpreted by the -answer- statement. Like the -store- command, the -storen- command terminates judging with a "no" judgment if no number or legal expression is found in the student's response. The -write- statements on lines 3 and 5 will be displayed if no number or only one number is entered, respectively. The student must enter another height specifying both feet and inches.

The -exact- Command

Occasionally you may want to require the student to type his answer in one specific way. In a lesson on expressing ratios using the a:b format, you would want the student's answer to be in exactly that form. The -exact- command is used for this (-exact- is a judging command).

```
unit    ratio
at      14Ø2
write   Express the ratio 3/5 using a colon.
arrow   16Ø1
exact   3:5
write   Good.
```

In order to match the tag of an -exact- command, the student's response must be the same, character for character, as the tag, including spaces and punctuation marks. With -exact- there is no mark-up of an incorrect response as there is with -answer-.

There is a conditional form of the -exact- command, -exactc-.

```
define  problem=v42
unit    setup
setperm 4
jump    ratios
*
unit    ratios
next    ratios
randp   problem
jump    problem,x,alldone,x
at      14Ø2
write   Express this ratio using a colon.
at      where+2
writec  problem↕↕↕3/5↕1/2↕4/5↕8/7↕↕
arrow   16Ø1
exactc  problem,,,3:5,1:2,4:5,8:7,,
```

A number is chosen by the -randp- and stored in the variable "problem". The item corresponding to this number is displayed, and the student's response is compared to the appropriate argument of the -exactc- tag. For instance, when the -randp- picks the number 3, the item 4/5 is displayed; and the student's response is compared to the tag 4:5. With the -answer- and -wrong- commands, punctuation marks are just taken as word separators. If punctuation marks need to be checked in the student response, use the -exact- or -exactc- commands.

Student Dialogs

At times you may want to allow such a wide variety of student responses that use of the -answer- command would be unwieldy. If you ask the student an open-ended question which requires him to answer in his own words, or conversely, if in your lesson the student is asking questions of PLATO, the number of "correct" responses will be huge. Two commands; -vocabs- and -concept-, allow you to specify three types of items: ignorable words, required words and their synonyms, and ideas or concepts which are anticipated student responses. The student's response will be judged "ok" if the idea of the student response is contained in one of the author anticipated responses (tag of the -concept- command) and if the words used in the student's response are part of the large vocabulary specified in the tag of the -vocabs- command. Here is a simple example:

```
unit    heating
vocabs  therm,<a,thermostat,that,which,is,device,it,the>
        (controls,regulates)
        (heat,temperature,furnace)
.
.
.
unit    question
at      8Ø5
write   What does a thermostat do?
arrow   1Ø1Ø
specs   bumpshift
concept controls temperature
```

Student responses like "It is a device that regulates temperature." or "It controls the heat." or "A thermostat regulates heat." are all judged "ok" by the single -concept- command.

The -concept- and -vocabs- commands are very useful in situations which require the student to describe a process, such as how to prepare specimens for a biology experiment; or situations in which the student needs to elicit information from PLATO, as in a lesson on medical diagnosis.

Note: This discussion of -concept- and -vocabs- is not intended to be complete. Rather, it is included here to alert you to more of the capabilities of PLATO for interacting with students.

9. More Display Features

Additional Features of -at- and -write-

We have seen that an -at- command is used to position text on the panel. The -at- command also sets up a left margin for succeeding lines of a continued -write- statement.

```
at       1Ø1Ø
write    When in
         the course
         of human events
```

The word "When" will begin at 1Ø1Ø, "the" will begin 111Ø, and "of" will begin at 121Ø. If there are more characters in the tag of the -write- than will fit on the line, the writing wraps around to the next line. The margin set by the -at- command still holds. The statements

```
at       1Ø55
write    When in
         the course
         of human events
```

will appear on the panel like this:

```
         When in
         the course
         of human e
         vents
```

The characters "vents" will not fit on line 12, so they are written starting at 1355.

Two consecutive -write- commands with no intervening -at- will cause the second -write- tag to begin immediately after the first. These statements

```
at       16Ø4
write    The cow jumped
write    over the moon.
```

will appear on the panel as:

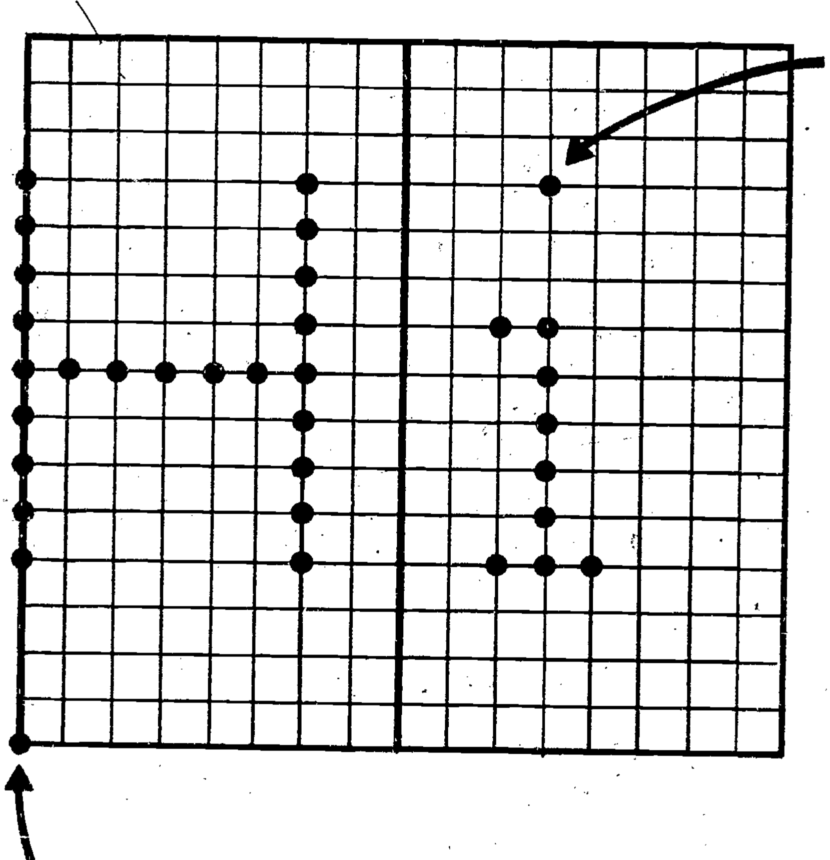
```
The cow jumpedover the moon.
```

If a space is written either after "jumped" or before "over" the sentence will be displayed with correct spacing.

A character location is a grid 8 dots wide and 16 dots high. The panel location specified by the tag of the -at- command, with either a fine or coarse grid format, refers to the lower left dot of the 8 by 16 character box and not the lower left dot of a standard character. Each character is positioned in the 8 by 16 box in a manner which considers the height and width of the character (for example, i is quite different from H). The characters displayed by the statements

```
at      200,150
write   Hi
```

are positioned in an 8 by 16 grid in figure 9.1.



location: 211,162

location: 200,150

Figure 9.1 Character grid, 8 by 16 dots

Writing in Different Modes

The -mode- command allows erasing characters from the display, or erasing an area of the panel before displaying new material in that area. The terminal has three possible modes: write, erase, and rewrite. The terminal is normally in write mode, but this may be altered by a -mode- command. The three possible tags of the -mode- command are "erase", "rewrite", and "write".

When a -write- command is executed and the terminal is in erase mode, the dots making up the characters are turned off (erased), instead of lit (written). Below is an example.

```
at      1612
write   This writing will disappear.
pause
mode    erase
at      1612
write   This writing will disappear.
mode    write
```

The sentence will be displayed, and when the student presses any key after the -pause- command, the sentence will be erased. You must return to write mode after a -mode erase- statement, or all subsequent writing will be invisible. When a new main unit is entered, PLATO automatically returns to write mode.

Rewrite mode causes the erasure of anything in the character space in question before new writing is displayed.

```
at      12Ø4
write   Write mode only "lights" the appropriate dots to
        form the characters.
pause
at      12Ø4
mode    rewrite
write   Rewrite mode erases a character space, then "lights"
        the dots corresponding to the new text writing.
```

The systems reserved word "mode" is set to -1 if you are in erase mode, Ø if in rewrite mode, and 1 if in write mode. The -mode- command may be used conditionally, e.g.

```
mode    v32,x,rewrite,write,rewrite,x
```

The "x" means that the mode presently in effect will not be changed.

A display may be erased by an -erase- command, an appropriate -mode- command, or by changing the main unit. In the last case, the entire panel is erased. This automatic erasure may be overridden by placing the statement

inhibit erase

in the main unit containing the display to be retained. When the student leaves a unit containing an -inhibit erase- statement, his main unit is changed, but all writing and drawing remains on the panel. If desired, -erase- or -mode- commands with appropriate tags can be used to erase parts of the display no longer needed.

In the example in figure 9.2, the student is in unit "title". The sentence "What is the product?" is written, and there is an immediate jump to unit "multiply". However, because of the -inhibit erase- statement in unit "title", the sentence is not erase when the student jumps to a new main unit (unit "multiply"). When the student finishes an item, a check is made (line 22 and unit "addone") and if he has completed 5 items without error he is jumped to unit "done". If he has fewer than 5 items correct on the first attempt, he continues with the unit. The problem is erased by "writing" it again in mode erase (lines 23 - 25), and the student's response is erased (lines 26 - 28). The student is jumped to unit "multiply" and is presented with another problem. The -inhibit erase- on line 9 prevents the erasure of the problem when the student is jumped to unit "done".

Another possible tag of the -inhibit- command is "arrow". The following statement will inhibit the display of the arrow.

inhibit arrow

All inhibits are cleared when a new main unit is entered or when an -inhibit- command without a tag is executed.

The -erase- command with either a one-argument or a two-argument tag has been discussed. An -erase- command with no tag causes a full screen erase. This is useful when you want to erase all display without changing the main unit.

```
*"first", "second", and "right" are defined in i.e.u.
unit    title
inhibit erase
at      91Ø
write   What is the product?
jump    multiply
*
unit    multiply
inhibit erase
randu   first,1Ø
randu   second,1Ø
at      1215
write   ⊲s,first⊳ × ⊲s,second⊳ =
arrow   where+2
ansv    first×second
do      ntries,x,x,addone,x
wrongv  first+second
write   That's addition.
no
write   Try ⊲s,first×second⊳
endarrow
jump    right-5,x,done,x
mode    erase
at      1215
write   ⊲s,first⊳ × ⊲s,second⊳ =
mode    write
at      where+2
erase   3Ø  $$erases student answer
jump    multiply
**
unit    done
at      242Ø
size    2
write   very good
size    Ø
*
unit    addone
calc    right←right+1
```

Figure 9.2 Units illustrating -inhibit-, -erase-, and -mode-

9.6

Figures Drawn from a Reference Location

The -rdraw- command is used to create drawings which need to be "moved", "sized", or "rotated". When using -rdraw- you must specify a location with an -at- command and all subsequent drawing is done with respect to that point. If no -at- command is given, the location of the last panel activity is used as the point of reference. Arguments in the tag of the -rdraw- statement can be given in either fine grid coordinates (dots) or coarse grid coordinates (lines and characters). Analogous to the -draw- command, points of the -rdraw- are separated by semicolons and fine grid x and y coordinates are separated by a comma. The arguments of the -draw- statement are actual panel locations, while the arguments of the -rdraw- are locations relative to the preceeding -at-.

Below is an -rdraw- statement which draws a triangle with the coarse grid location 1632 as the point of reference. Figure 9.3 shows how this triangle appears to the student.

```
        at      1632
        rdraw   4Ø4;Ø;136,-1;4Ø4
```

The first (4Ø4), second (Ø), and fourth (4Ø4) points are in coarse grid coordinates, and the third point (136,-1) is a fine grid coordinate. All points are relative to the reference (1632) specified by the preceeding -at- statement. To find the actual panel locations specified by -rdraw- and the preceeding reference point, one sums the reference location with each argument in the tag of the -rdraw-. This is summarized for this example in figure 9.4.

All -rdraw- commands can be "sized" by placing a -size- statement before the -rdraw- statement.

```
        at      1632
        size    2
        rdraw   4Ø4;Ø;136,-1;4Ø4
```

The -at-, -size-, and -rdraw- commands above produce a triangle twice as large as the triangle in figure 9.3. Each number in the tag of the -rdraw- is (in effect) multiplied by 2.

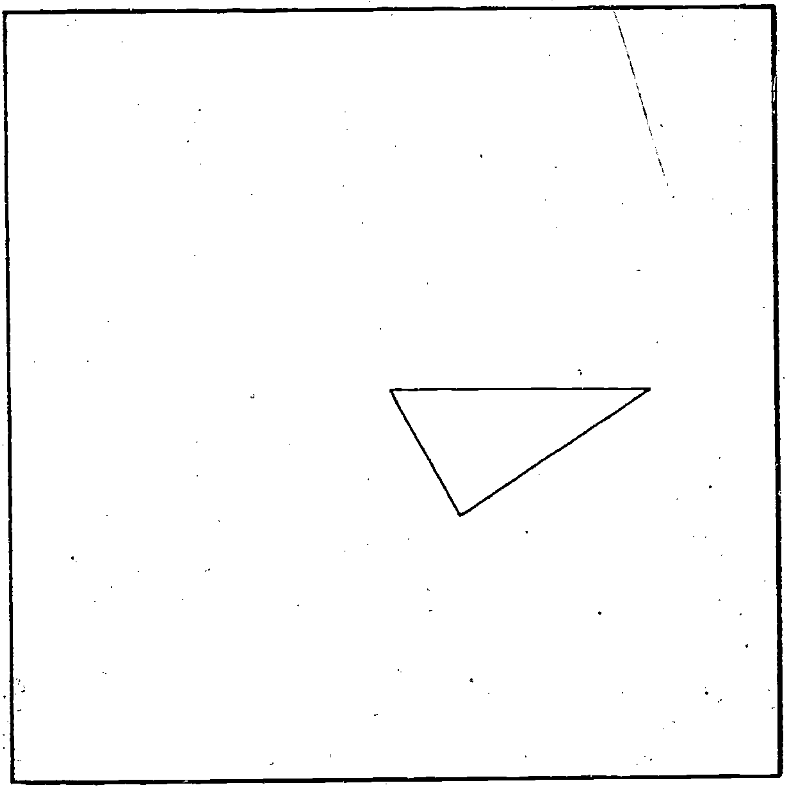


Figure 9.3. A triangle with -rdraw-

point of reference as specified by -at- is 1632

(fine grid is 248,256)

```
rdraw   4Ø4;Ø;136,-1;4Ø4
       /     | \_____/  \
   point 1   |  point 3   point 4
             |
          point 2
```

point	tag	coarse or fine grid	location on display panel
1	4Ø4	coarse	1632 + 4Ø4 = 2Ø36
2	Ø	coarse	1632 + Ø = 1632
3	136,-1	fine	{248,256} + {136,-1} = 384,255
4	4Ø4	coarse	1632 + 4Ø4 = 2Ø36

Figure 9.4 Actual location of points specified by -rdraw- and -at-

Figure 9.5 shows the same -rdraw- statement used in a -do- loop with the degree of rotation increased on each iteration of the loop, creating a pinwheel. (Iterative -do- statements are discussed in Chapter 11.) The TUTOR code to produce the pinwheel is shown below.

```
unit    pinwheel
do      triangle,v2⇐1,12
*
unit    triangle
rotate  3Ø*v2
at      1632
rdraw   4Ø4;Ø;136,-1;4Ø4
```

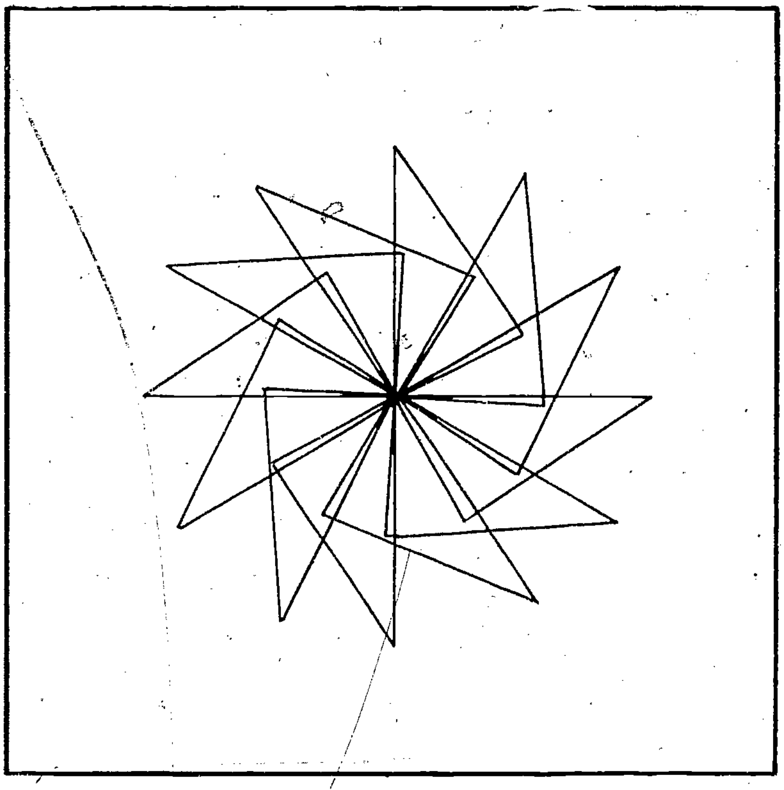


Figure 9.5 Pinwheel <u>via</u> -rdraw- and -do-

Charts and Graphs

While it is possible to create graphs using only -at-, -write-, -draw-, -rdraw- and -circle- commands, TUTOR has commands which make it easy to construct charts, label axes with descriptive legends or borders, and to construct bar or pie charts using any characters desired. Graphs with scaled rectilinear axes, semi-log axes, or log-log axes may be drawn easily. Functions may be plotted pointwise or continuously, in scaled coordinates relative to the origin. Polar functions may also be plotted. These commands are described in detail in the graphing section of lesson "aids" (refer to Chapter 13).

Custom Made Characters

An author may need special symbols relevant to the subject matter of his lesson, but which do not exist in the character set built into the terminal. Some examples of this are the Cyrillic alphabet, phonetic symbols, chemical symbols, or even pictures of animals.

In addition to the 126 characters built into the terminal (a, b, c, 1, 2, 3, A, B, C, etc.), there are 126 more character slots in the terminal which can contain characters designed by an author. Each of these characters is associated with one of the keys on the keyset, but to access the special character you must first press FONT (the shifted key to the right of ERASE) and then the standard key. Subsequent key presses are in the "alternate font"; pressing FONT again returns you to the regular font.

You may create one or more special characters by creating a new block in a lesson and designating it as a "charset" block. When you enter the charset block you are asked what key your character is to be associated with. Then you design your character by moving a cursor on an enlarged grid representing the character space. You may inspect the new character in actual size before it is stored. You may modify characters or add new ones to the set at any time.

The character set need not be part of the lesson using the special characters; it may be in any lesson. To associate a character set with a lesson for student use, place a -charset- command in the lesson:

```
charset lesson,blockname
```

The "lesson" is the name of the lesson in which the character set is stored, and "blockname" is the name of the blocks containing the character set.

The -charset- command should be placed in the i.e.u. so that the character set will be loaded into the terminal even if a student enters the lesson at a restart unit (see Chapter 1Ø). Below is a typical i.e.u. containing a -charset- command.

```
at       912
write    LOADING CHARACTER SET
         Please be patient --
         Loading takes about 17 seconds
charset  phonetics,ipa
erase
```

The -write- command informs the student of a short delay. The process takes a few seconds and without this message the panel would be blank. The tag of the -charset- command designates the lesson in which the character set is found, in this case lesson named "phonetics", and the block name of the character set, "ipa". The -erase- command erases the panel after the character set has been loaded, preventing the "LOADING . . ." message from being superimposed on display material.

A character set remains in the terminal even after the lesson which uses it is no longer in use by students or authors. Thus you may frequently find a terminal containing a character set other than the one you want to use. There are three ways to load the desired character set into the terminal. You may condense a lesson containing the proper -charset- command; you may edit the charset block and press the NEXT key; or you may type, on the author mode display, the word "charset" (without quotes), press NEXT, type the name of the lesson containing the character set, press NEXT, and then type the block name of the character set.

While editing your lesson you may switch between the built-in characters and the -charset- characters by pressing the FONT key. The same is true for the student working in the lesson.

Animation using special or regular characters is possible. Using the iterative -do- statement (discussed in Chapter 11) and rewrite mode, a character may be moved across the panel.

```
unit    animate
mode    rewrite
do      moving,index←1ØØ,4ØØ
unit    moving
at      index,2ØØ
write
```

The character, in this example a car, moves across the panel one dot at a time. If any character is designed with its leftmost column of dots blank, and rewrite mode is used, the old position will be erased as the character is written at the new position. This allows a "continuous" movement, one dot at a time, on the plasma panel. If the left-most two columns were blank, the character could be advanced two dots at a time, and so forth. The -size- and -rotate- commands have no effect on special character sets.

The Micro Option

A string of characters (standard, special, or a combination of the two) can be assigned to a single key. For example, assume an author has designed the character for a grave accent mark (ˋ) and assigned it to the key g. For the student to type the French word "là" he would have to type "l, a, backspace, FONT, g, FONT". But the author may create a "micro table" which assigns the character string "backspace, FONT, g, FONT" to the letter "g". Then the student may produce the entire character string (the properly placed ˋ) by striking the MICRO key and then g.

To create a micro table, add a new block in your lesson and designate it as a micro block. Specify the key which is to produce the string of keys, and then specify the character string. For a micro table to be available in student mode, the lesson must contain the statement

```
micro   lesson,blockname
```

where "lesson" is the name of the lesson containing the micro block and "blockname" is the name of the micro block. As with charsets, the micro blocks need not be in the lesson which uses them. A convenient location for the -micro- command is in the i.e.u.

Color Microfiche

A terminal may be equipped with a slide selector, allowing slide images to be back projected through the plasma panel. Up to 256 colored slides, in microfiche format, are randomly addressable by the computer. This allows the superposition of graphically displayed text and drawings on color or black and white slides. The -slide- command turns on the slide projector and displays the slide specified in the tag of the -slide- command. Information on the preparation and use of slides is available in the slides section of lesson "aids" (refer to Chapter 13).

Touch and Audio

Auxiliary equipment available includes a touch panel, which allows the student to touch the panel instead of typing his input on the keyset. The touch panel has 256 individual addressable positions.

A random access audio device, under computer control, allows playing any one of more than 4ØØØ audio messages. Up to 21 minutes of audio messages may be recorded on a random access audio disc.

Cover Design

The design on the cover of this manual was made using sized writing and figures drawn with respect to a point of reference. The TUTOR code which produces the design on the cover is presented in figure 9.6. This is just an example of some of the display capabilities discussed in this and previous chapters.

```
define  degree=v1
*
unit    cover
do      sector,degree←Ø,36Ø,2Ø
at      19Ø,273  $$ erase center of figure
erase   15,5
at      2Ø3,268
write   INTRODUCTION
            to
size    2.5      $$ sized writing
at      2Ø2,213
write   TUTOR
at      2Ø3,213  $$ write 3 times to make thick letters
write   TUTOR
at      2Ø2,214
write   TUTOR
size    Ø
at      3Ø48
write   James Ghesquiere
        Celia Davis
        Charlene Thompson
*
unit    sector
rotate  degree
at      25Ø,25Ø
rdraw   Ø,Ø;15Ø,2ØØ;8Ø,12Ø;-4Ø,6Ø;Ø,Ø
```

Figure 9.6 TUTOR code for cover design

1∅. Special Branching Features

Using the TERM Key

We have seen that a student may initiate a branch in the lesson by pressing a HELP-type key, if the unit he is in contains a help-type command. The TERM key (please note that TERM is a shifted key) and its associated command -term- also allow the student to move to a different section of the lesson. However there is a major difference between -term- and -help-. The -term- command is placed in the unit the student is to be sent to, not the unit he is in now. For example:

```
unit    quests
next    abacus
at      5∅5
write   Following are some questions about the history of
        computing machines.
        If at any time you would like to review, press
        TERM and type the word "review" (without the quotes).
*
unit    abacus
.
.
.
unit    history
term    review
at      5∅3
write    6∅∅ B.C.    Abacus comes into use
        1642.A.D.    Pascal builds first adding machine
        .
        .
        .
        .
        1959         PLATO I
at      3∅1∅
write   Press BACK to return.
end     help
```

In any unit in the lesson, when the student presses TERM, the phrase "What term?" and an arrow appear at the bottom of the panel. If he types "review" and presses NEXT, he will proceed to unit "history". From unit "history" pressing BACK or BACK1, or pressing NEXT (since unit "history" contains an -end help- statement) will return him to the unit from which the TERM key was pressed (just as with HELP-type keys). Unlike the HELP-type keys, the TERM key is active anywhere in a lesson. If the "term" the student enters exactly matches the tag of a -term- command, he will be taken to the unit containing that -term- statement. If his "term" does not match the tag of any -term-, his input is erased, along with the "What term?" message and the arrow, and the TERM keypress is ignored.

Any number of -term- commands may appear in a lesson, but experienced authors have found that the use of more than one or two -term- statements tends to confuse the student. The student must be informed early in the lesson of available terms. Some authors use only one -term- statement:

```
        term    index
```

in their lesson. The TERM key takes the student to a page listing all the topics in the lesson, and the student chooses a topic. Of course such an approach assumes a lesson structure which may not be appropriate in some subjects.

Altering the Base Unit

If the student presses BACK or BACK1 from a unit in a sequence reached by TERM or a HELP-type branch, he is returned to his base unit. However, if there is a -back- or -back1- statement in the student's current main unit and the corresponding key is pressed (BACK or BACK1), the student is branched to the unit named in the tag of that -back- or -back1- statement and the help sequence is continued. In other words, the execution of a -back- or -back1- statement has precedence over the default function of a BACK or BACK1 keypress terminating a help (or term) sequence.

The base unit may be altered by the author, if desired, with a -base- command. The -base- command with no tag clears the base pointer so the student is no longer in a help sequence and therefore has no base unit. The -base- command with no tag changes the current help sequence to a main lesson sequence. When using a -term- command in a topic index unit, for example, it is usually desirable to clear the base unit pointer so the student starts a new main lesson sequence.

```
unit    topics
base
term    index
at      8Ø3
write   Press the number . . .
.
.
store   choice
no
jump    choice-2,unita,unitb,etc.
```

Without the -base- command a student could conceivably be in a help sequence with his base pointer set, reach unit "topics" by the TERM key, proceed through a section of the lesson, enter another HELP sequence, then press BACK1 and be returned to some unit relevant to the section he was in before he pressed TERM and entered unit "topics".

You may wish to devise a structure whereby after a help sequence has ended, the student is returned to a unit preceding the one in which he pressed HELP, as in the units below.

```
unit    geoma
data    geohelp
next    geomb
.
.
.
unit    geomb
data    geohelp
next    geomc
.
.
.
.
unit    geohelp
base    geoma
.
.
.
end     help
```

If the student presses DATA from either unit "geoma" or "geomb", he is sent to unit "geohelp". But in either case he is returned to unit "geoma" after completing unit "geohelp". If the student enters the help sequence from unit "geomb", the base unit pointer is set to "geomb", but the -base geoma- overrides this and sets the base unit pointer to "geoma". The base command with a unit named in the tag makes the named unit the new base unit.

Finishing and Restarting a TUTOR Lesson

How does one end a TUTOR lesson? The statement

```
        end     lesson
```

may be placed at the logical end of the lesson. When this statement is encountered the student is returned to the "Welcome to PLATO" display. The author could also insert a -write- statement in the last logical unit telling the student he has finished the lesson and instructing him to press STOP1 to exit.

Often a student will not complete an entire lesson in one session. If the lesson contains no commands to the contrary, the student will begin in the first unit of the lesson when he signs on again. To put this sequencing under the control of the author, the -restart- command is available. There are three forms of the -restart- command:

```
        restart
        restart unitname
        restart lesson,unitname
```

When the student signs on after having signed off without completing a lesson, he begins in the lesson and unit specified by the -restart- command last encountered. If a -restart- with no tag has been encountered, he begins in the unit containing the -restart- command. If a -restart- command naming a unit or a lesson and unit has been encountered, he begins in the named lesson and unit.

A "finish unit" may be designated in a lesson. This is a unit which is executed whenever a student leaves a lesson by means of STOP1. The command has the form

```
        finish  unitname
```

The statement should be placed in the i.e.u. to insure its being in effect for students who enter the lesson in a "restart unit". The finish unit may not contain any -write- or -show- commands, but it is a convenient place to do any calculations, etc., which must be done to insure correct sequencing when the student signs on again. Note: The finish unit is not executed when a student exits because of an -end lesson- statement.

11. Looping

The Iterative Form of the -do- Command

The apple was dropped in exercise C, figure 3.5, by having a series of -do- commands, one -do- command for each movement of the apple. The iterative form of the -do- command will repeatedly -do- a unit.

```
        do      move,v37←1,7
```

Unit "move" is executed seven consecutive times as the variable v37 is incremented from one to seven. The general form of the iterative -do- command is:

```
        do      unit,index←start,end
```

The unit named in the first argument of the tag is executed until the current value of "index" is greater than "end". The value of "index" is increased by 1 each time the loop is completed. If the increment is to be different from 1, it is specified as a fifth argument in the tag of the -do- command.

```
        do      unit,index←start,end,increment
```

The "index" must be a variable while "start", "end", and "increment" may be constants, variables, or expressions. A negative value for the increment will cause the loop to go "backwards".

```
   1    unit    doit
   2    calc    radius←2Ø
   3    do      circles,index←1,3
   4    at      3Ø2Ø
   5    write   Here are some circles.
   6    *
   7    unit    circles
   8    circle  radius,256,256
   9    calc    radius←radius+2Ø
```

Three circles will be drawn by the above units. After each circle is drawn (line 8), "radius" is incremented by 20 dots (line 9), so the circles will have radii of 20 dots, 40 dots, and 60 dots, respectively. A flow chart, figure 11.1, illustrates the actions of these units.

Unit "circles" is done three times. The variable "index" is incremented automatically by TUTOR and compared to the ending value each time the loop is done. If ten circles were needed instead of three, only the last argument for the -do- statement, line 3, would need to be changed (from 3 to 10).

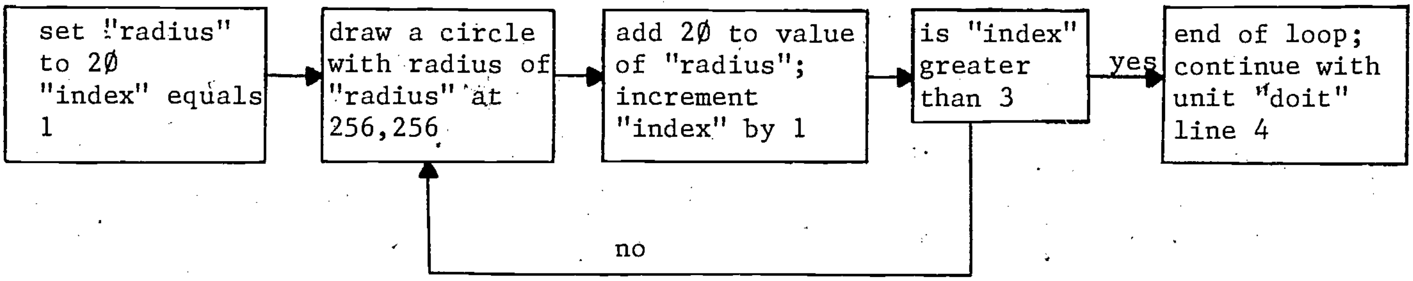

set "radius" to 2∅ "index" equals 1
draw a circle with radius of "radius" at 256,256
add 2∅ to value of "radius"; increment "index" by 1
is "index" greater than 3
yes
end of loop; continue with unit "doit" line 4
no

Figure 11.1 Flow chart for iterative -do-

Figure 3.3 in Chapter 3 displayed a table of data by executing the same unit several times. The eight -do- statements in figure 3.3 are replaced by an iterative -do- statement in figure 11.2. The iterative -do- is also used to set and increment the location of table entries by using "place" as the index of the -do-. The TUTOR code in both figures 11.2 and 3.3 will create the display in figure 3.4.

The -join- command may also be used in an iterative manner. The form of the repeated -join- command is exactly the same as the repeated -do- command. The -join- and -do- commands differ only in that -do- is a regular command only, while -join- is executed in regular and judging state.

```
unit    table
at      4Ø7
write   Table of radii and circumferences.  Press NEXT for
        each entry to be displayed.
draw    1224;2424;2462;1224;skip;1424;1462;skip;1243;
        2443
at      23Ø,296
write   radius
at      36Ø,296
write   circumference
calc    place←1628  $$ point of reference
        radius←1  $$ set initial radius
do      circ,index←1,8
*
unit    circ
pause
circle  radius,9Ø,23Ø
at      place
show    radius
at      place+2Ø
show    2π×radius
calc    place←place+1ØØ  $$ move down 1 line
calc    radius←radius+1Ø  $$ increase the radius
```

Figure 11.2 Revised figure 3.3 using iterative -do-

The -goto- Command

The -do- and -join- commands are used to attach units to a main unit. The -goto- command is also used for this purpose. We have seen that when a -do- command is executed, PLATO treats the contents of the attached unit as if they appeared right in the main unit. Commands below the -do- or -join- are processed after the statements in the attached unit are processed. Executing a -jump- command, on the other hand, causes an immediate branch to a new main unit, and commands below the -jump- are not executed.

When a -goto- command is executed, the main unit is not changed, but commands below the -goto- are not done. The units in figure 11.3 illustrate how -goto- can be used to take a student "out" of a unit, yet not disrupt his path through the material. Since -goto- is a regular command, it will be done only if the student's response matches the -answer- tag on line 11 of figure 11.3. If so, the student will be branched to unit "blithe". When he presses NEXT he will proceed to unit "stories". The pointer set by the -next- command in unit "novels" remains in effect in unit "blithe", because unit "blithe" is not a new main unit. A student whose response does not match the tag of the -answer- command on line 11 will not see unit "blithe"; he continues in unit "novels", seeing the portion of the unit below line 13. When he completes the unit and presses NEXT he will proceed to unit "stories".

```
 1   unit     novels
 2   next     stories
 3   at       81Ø
 4   write    What Hawthorne novel is concerned with the conflict
 5            between social tradition and a tistic self-expressio
              n?
 6   arrow    11Ø8
 7   specs    okspell,bumpshift
 8   wrong    <the> marble faun
 9   write    The main conflict in that novel is between
1Ø            different forms of artistic expression.
11   answer   <the> blithedale romance
12   goto     blithe
13   answer   <the> scarlet letter
14   write    Yes.  The main characters' dress reinforces the
15            conflict symbolically.
16   at       181Ø
17   write    The magistrates are always dressed in grey or black,
18            with rigid hats and stiff shirts, while Pearl is
19            always dressed in bright colors.
2Ø   **
21   **
22   unit     blithe
23   at       181Ø
24   write    The contrast is embodied in the differences between
              the
25            two heroines.
26   **
27   unit     stories
     .
     .
     .        more TUTOR code
     .
     .
```

Figure 11.3 Code for literature units

The -goto- command may also be used in a conditional form. After an initial -write- statement, unit "history" in figure 11.4 is completed by either asking a question, going to unit "lee", or going to unit "grant" depending on the value of "errors". This process is summarized in figure 11.5.

```
1    unit     history
2    at       712
3    write    During the Civil War, . . .
4    goto     errors,x,lee,grant,grant,x
5    arrow    1122
6    specs    okextra,bumpshift
7    next     grant
     answer   ...
     .
     .
     .
     unit     grant
     .
     .
     .
     unit     lee
     .
     .
     .
```

Figure 11.4 -goto- used conditionally

value of "errors"	complete unit "history" with:
negative	a question, lines 5 through end of unit "history"
zero	unit "lee"
one	unit "grant"
two	unit "grant"
more than two	a question, lines 5 through end of unit "history"

Figure 11.5 Summary of effect of conditional -goto- in unit "history"

The -goto- Command and the "q" Branch

The -goto- command is usually used in the conditional form. The current main unit in the statement below is continued or completed with unit "one", "two", or "three".

```
goto     counter,x,two,x,one,three,x
```

To stop the processing of regular commands in the current main unit, insert the argument "q" (for quit) in the corresponding position of the -goto- tag. The following tag is summarized in figure 11.6

```
goto     counter,x,two,q
```

The "q" or quit branch can effectively be used as part of a calculational looping or branching sequence. When the "q" branch is used with function key branching commands, it means "clear" that pointer for the current main unit.

```
goto     counter,x,two,q
```

value of "counter"	tag	interpretation
negative	x	fall through to next TUTOR command
zero	two	complete main unit with unit "two"
one and greater	q	complete main unit with this -goto- statement, do NOT fall through to next TUTOR command

Figure 11.6 Summary of a conditional -goto- statement using "q"

12. Additional Ways to Use Variables and Do Calculations

Storing and Showing Characters

Numerical expressions are evaluated and placed into variables with the -store- command. Characters can be placed into variables named in the tag of the -storea- command in an analogous manner. The -showa- command displays in alphanumeric form the contents of the variable named in its tag.

```
unit    name
at      162Ø
write   What is your name?
arrow   2Ø25
storea  student  $$ "student" is defined previously
ok
write   Thanks,
showa   student
```

Ten alphanumeric characters can be placed in each TUTOR variable. The -storea- and -showa- commands are designed for use with alphanumeric characters, whereas the -store-, -show-, and -showt- commands are designed for use with numbers.

Each TUTOR variable can be displayed as a number with the -show- or -showt- command, or in "scientific notation" with the -showe- command. The variable can also be displayed as a group of letters with the -showa- command or as an octal number with the -showo- command. Even though there are several formats in which the same TUTOR variable can be displayed, in general only one format is appropriate for the information stored in a given variable. In other words, if letters are stored in v23, it is only appropriate to display that information with a -showa- command, and not a -show- or -showe-.

Conditional Calculations

In addition to the -calc- command, there are two commands which perform one of a list of calculations based on some condition. The -calcc- command (pronounced: calc see) will perform one of several different calculations based on the value of an index expression, whereas the -calcs- command (pronounced: calc ess) will assign to the same variable one of several different values based on the value of an index expression. The format of the

-calcc- and -calcs- commands follows the usual TUTOR conditional format: index expression, negative argument, zero argument, through the positive integer arguments.

The index expression for a -calcc- or -calcs- may be a single variable or a complex arithmetic expression containing several variables and operators. The statement

calcc   v1+3v4,v2←2Ø,v14←(v12)^2,v37←v2Ø+3v6,,v14←Ø,,

will evaluate the index expression (v1+3v4) and execute the calculation in the corresponding position of the tag of the -calcc- as summarized in figure 12.1. If no calculation is to be performed (a "fall through" situation) for the conditional calculation commands, no argument is entered in the appropriate position of the -calcc- (or -calcs-) tag. In the above example, no calculation is performed when the index expression (v1+3v4) equals 2 or when the index expression is 4 or larger.

calcc   v1+3v4,v2←2Ø,v14←(v12)^2,v37←v2Ø+3v6,,v14←Ø,,

value of "v1+3v4"	calculation to be performed
negative	v2←2Ø
zero	v14←(v12)^2
one	v37←v2Ø+3v6
two	none
three	v14←Ø
four and greater	none

Figure 12.1 Conditional calculation command, the -calcc-

To assign one of several different values to the same variable you can use the -calcs- command. The statement

calcs   locate+3,history←8,4,history+1,Ø,6,,history-1 causes different values to be assigned to the defined variable "history" based on the value of the expression "locate+3". Figure 12.2 summarizes the assigned value of "history" as a function of "locate+3".

```
calcs   locate+3,history←8,4,history+1,Ø,6,,history-1
```

value of "locate+3"	value assigned to "history"
negative	history←8
zero	history←4
one	history←history+1
two	history←Ø
three	history←6
four	no change to "history"
five and greater	history←history-1

Figure 12.2 Example of -calcs- statement

For a molecular weight drill in chemistry, the -calcs-, -writec-, and -ansv- can be used very effectively. The use of conditional commands and defined variables allows an easy expansion of the questions presented by unit "molwts". By adding arguments to lines 5 and 7 of this unit, one could have a drill with ten or more questions of the same or increasing level of difficulty. By including the "sampling without replacement" method of choosing the value of "problem" (i.e., -setperm-, -randp-, -remove-, and -modperm- commands), a drill could be written which automatically reviews non-mastered items.

```
 1   unit     molwts
 2   *problem and molwt are previously defined and initialized
 3   at       1212
 4   write    What is the molecular weight of  $$ leave 1 space
 5   writec   problem,,,$H_2O$,$HNO_3$,$CaCl_2$
 6   write    ?
 7   calcs    problem,molwt←,,,18,63,111
 8   arrow    1819
 9   ansv     molwt,1%
1Ø   no
11   write    Refer to the Periodic Table
12            and sum the Atomic Weights.
```

Logical Operators

By combining the conditional format of TUTOR commands and the power of logical operators, a new dimension is added to the index expression of all conditional commands. The basic logical operators available are: =, ≠, <, ≤, >, ≥, which mean equal, not equal, less than, less than or equal to, greater than, and greater than or equal to, respectively. If a logical expression is true, it has a value of -1; if false, a value of Ø. For the -do- statement

```
do      correct=3,out,more
```

unit "out" is attached if "correct" equals 3, and unit "more" is attached if "correct" has any value different from 3. The statement

```
goto    errors>4,review,x
```

means the current main unit will be completed with unit "review" if "errors" is greater than 4. If "errors" is equal to or less than 4 (i.e., if the expression is false), the logical expression has a value of Ø and one "falls through" to the following TUTOR statement because of the x.

Logical operators may appear in any type of calculation or expression. The -calc- statement

```
calc    radius←5Ø-37×(3=y)
```

gives "radius" a value of 87 if "y" is equal to 3. If "y" is not equal to 3 (if the expression is false), "radius" is assigned the value of 5Ø; see figure 12.3.

using the expression:	50-37×(3=y)
if y=3	if y≠3
50-37×(3=3)	50-37×(3=4)
50-37×(true)	50-37×(false)
50-37×(-1)	50-37×(Ø)
50+37	50-Ø
87	50

Figure 12.3 Evaluating an expression containing a logical operator

The combination of logical expressions is possible with and and or. For the expression (3>y and 6=x) a true value (-1) is obtained if 3 is greater than "y" and if 6 is equal to "x". Under all other conditions a Ø (false) is obtained. For the expression (3>y or 6=x) a true value is obtained if either 3 is greater than "y", or if 6 is equal to "x", or if both conditions are true. The combining operations, and and or, make sense only when used with logical expressions. The expression "radius and 41" is meaningless and will produce unpredictable results. (For those interested, there are also operators for the following bit manipulations: mask, union, difference, and shifts within a word. All bit manipulations must be done with the integer representation of TUTOR variables. Refer to lesson "aids" for details; see Chapter 13.)

Segmented Variables

As a student completes a portion of a lesson, some evaluation of progress may be made and the need for review may be decided. Often one need only set a "flag" indicating if review is needed or not. Each student variable is composed of 6Ø "bits", each of which can be set to either Ø or 1. When only a yes or no decision needs to be made (e.g. does this student need review?) it may be wasteful to set aside an entire student variable when only one bit would suffice.

The ability to segment defined variables provides a very easy method of storing many pieces of information per variable, instead of only one piece of information per variable. If one considers the situation of reviewing portions of a lesson, a value of Ø may mean "no review" and a value of 1 may indicate review is needed. This information could be stored for 6Ø different portions of a lesson(s) in a single variable by including the following -define- tags with your -define- set:

```
define  yourset
        segment,review=v1,1
        norev=Ø,yesrev=1  $$ you may define constants
```

The student variable "v1" is segmented into 6Ø pieces (or bytes), and each piece is one bit long. Each bit can have a value of Ø or 1, so each bit can "remember" if there should be review (if the bit equals 1) or if no review is needed (if the bit equals Ø). The size of the byte, one bit in this example, is indicated by the last argument of the segment of the -define- statement.

The 43rd segment is referenced by the format: review(43). For example:

```
calc    review(43)←1  $$ 1 means yes; review is needed
```

A defined segment may be used as an index expression in a conditional form of a command and as part of a logical statement.

```
goto    review(12),x,ok,needed
goto    review(12)=yesrev,needed,ok  $$ yesrev is defined as 1
```

These two -goto- statements are equivalent. Unit "ok" is attached to the main unit if the twelfth segment of "review" equals Ø; if the 12th segment of the defined variable "review" equals 1, unit "needed" is attached.

The byte size may be as large as 59 bits, but a byte size of more than 3Ø bits is no real savings of storage space. The following statements are an example of a variable segmented into ten pieces, where each piece is 6 bits and can contain an integer as large as 63. The third segment is assigned the value of 22, and the value is displayed.

```
define  yourset  $$ all defined variables should be in the i.e.u.
        segment,errors=v14,6
*
calc    errors(3)←22
at      22Ø
write   The third error segment= ⟨s,errors(3)⟩.
```

Only integers can be stored in segmented variables. Non-integral numbers are represented by the computer in a format which requires an exponential, sign, and base portion and therefore cannot be directly stored in a "segment" of a variable as an integer can be stored. Figure 12.4 summarizes the maximum integer value, bits required, and the number of segments available for each 6Ø bit TUTOR variable which is segmented. If negative and positive numbers are stored in segmented variables, one bit per segment must be reserved for the sign. This discussion is intended to be only an introduction to the possible uses of segmented variables; refer to lesson "aids" (see Chapter 13) for details.

maximum integer value	bits per segment	segments per TUTOR variable
1	1	6Ø
3	2	3Ø
7	3	2Ø
15	4	15
31	5	12
63	6	1Ø
127	7	8
255	8	7
511	9	6
1Ø23	1Ø	6
4Ø95	12	5
32767	15	4
$2^{20}-1$	2Ø	3
$2^{30}-1$	3Ø	2

Figure 12.4 Segmented TUTOR Variables

Defined Functions

Many common functions are built into the TUTOR language. These systems defined functions are listed in figure 12.5. Functions may also be specified in the tag of the -define- command. Some examples are provided in figure 12.6 and illustrated in figure 12.7. Defined functions may be part of any calculation or condition. When encountered by TUTOR a defined function is placed in parentheses and is replaced by its "defined" meaning.

function	description
abs(x)	absolute value of x
int(x)	integer part of x
frac(x)	fractional part of x
round(x)	round x to the nearest integer
sqrt(x)	positive square root of x
log(x)	base 1Ø logarithm of x
ln(x)	base e logarithm of x
exp(x)	raise e to the power of x
sin(x)	sine function (x in radians)
cos(x)	cosine function (x in radians)
arctan(x)	arctangent function (x in radians)
bitcnt(x)	number of bits set in variable
not(x)	inverts logical truth values

Figure 12.5 Defined systems functions

You may also define your own functions. A function specified in the tag of a -define- command may involve a previously defined quantity on the right side of the equal sign, as on lines 3-5 of figure 12.6. The argument of the function on the left side of the equal sign is really a "dummy" argument ("r" on lines 1 and 2, "dummy" on line 4 of figure 12.6). Thus if "r" or "dummy" were previously defined, these functions of "r" or "dummy" would be rejected by TUTOR. A defined function may also contain an assignment arrow (line 5, figure 12.6).

As illustrated in figure 12.6, defined variables (line 3), defined functions (lines 1, 2, 4, and 5), segmented variables (line 6), and defined characters (line 7) may all be part of the same -define- statement. This discussion is only an introduction to the possible uses of defined functions; refer to lesson "aids" (see Chapter 13) for details.

```
1    define   circ(r)=2πr  $$ circumference of a circle
2             vol(r)=4πr^3/3  $$ volume of sphere
3             q=v6,review=v7,result=v8
4             funct(dummy)=dummy^2-3q
5             assign=(result←result+2Ø)
6             segment,time=v126,6
7             yes=1,no=Ø
```

Figure 12.6 Functions specified by -define-

```
TUTOR statements using                          value of "result"
-define- of figure 12.5                         after -calc-

calc    result←circ(5)                          1Øπ
calc    result←3*vol(2)                         3*4π2^3/3=32π
calc    result←funct(8)   $$ q=1Ø               64-3Ø=34
calc    assign  $$ effect is to add 2Ø to result
calc    result←int(3.7)                         3
calc    result←round(3.7)                       4
calc    result←log(2Ø)                          1.3
```

Figure 12.7 Examples of defined functions

13. What NEXT?

When you finish the material in lesson "introtutor", the chapters in this book, and the exercises, you will have completed this introductory TUTOR training program. This chapter offers some suggestions to guide you in your future activities with PLATO.

Developing Lessons

You will be most successful if you choose a short topic for your first lesson. This will help you to determine the feasibility of using PLATO for your subject matter. After your first lesson is written, you may want to ask some of your colleagues to use your lesson as a student. They may offer useful criticism about content and presentation, and will also uncover any program errors you may have missed.

After making revisions based on your colleagues' suggestions, arrange to have 5 or 1ø students use your lesson. You should be present at these sessions to observe the students' reactions to the lesson and to see whether the lesson fulfills your educational goals. The students may offer more suggestions for revision. You should use PLATO's automatic data collection features while testing the lesson. (Refer to "aids" for a complete description of data collection features.)

After your first topic is complete, you may want to write a few more modules and test them in the same way. Then you can decide which parts of your course should be coordinated with PLATO.

Learning More TUTOR

Lesson "aids", available in student mode, is an on-line reference manual of the TUTOR language. It contains overveiws of all areas of TUTOR, descriptions of all TUTOR commands, summary lists of commands and systems reserved words, and information on lesson design and implementation.

Before working on some new aspect of your lesson, you should read a relevant portion of "aids". Write test units experimenting with the commands that are new to you until you feel comfortable with them. This experimentation may require reading several sections of "aids" and talking with TUTOR consultants. As you do more testing, you will understand more of the capabilities of the TUTOR language. An advanced TUTOR manual, entitled "The TUTOR Language" is available as well as several other publications. All publications may be purchased by contacting PLATO Publications, Computer-based Education Research Laboratory, University of Illinois, Urbana, Illinois 61801.

Keeping Current

The lesson "notes" contains many different kinds of notes as the index display from lesson "notes" indicates, figure 13.1. The "New System Features and "General Interest Notes" sections let you maintain a current knowledge of TUTOR.

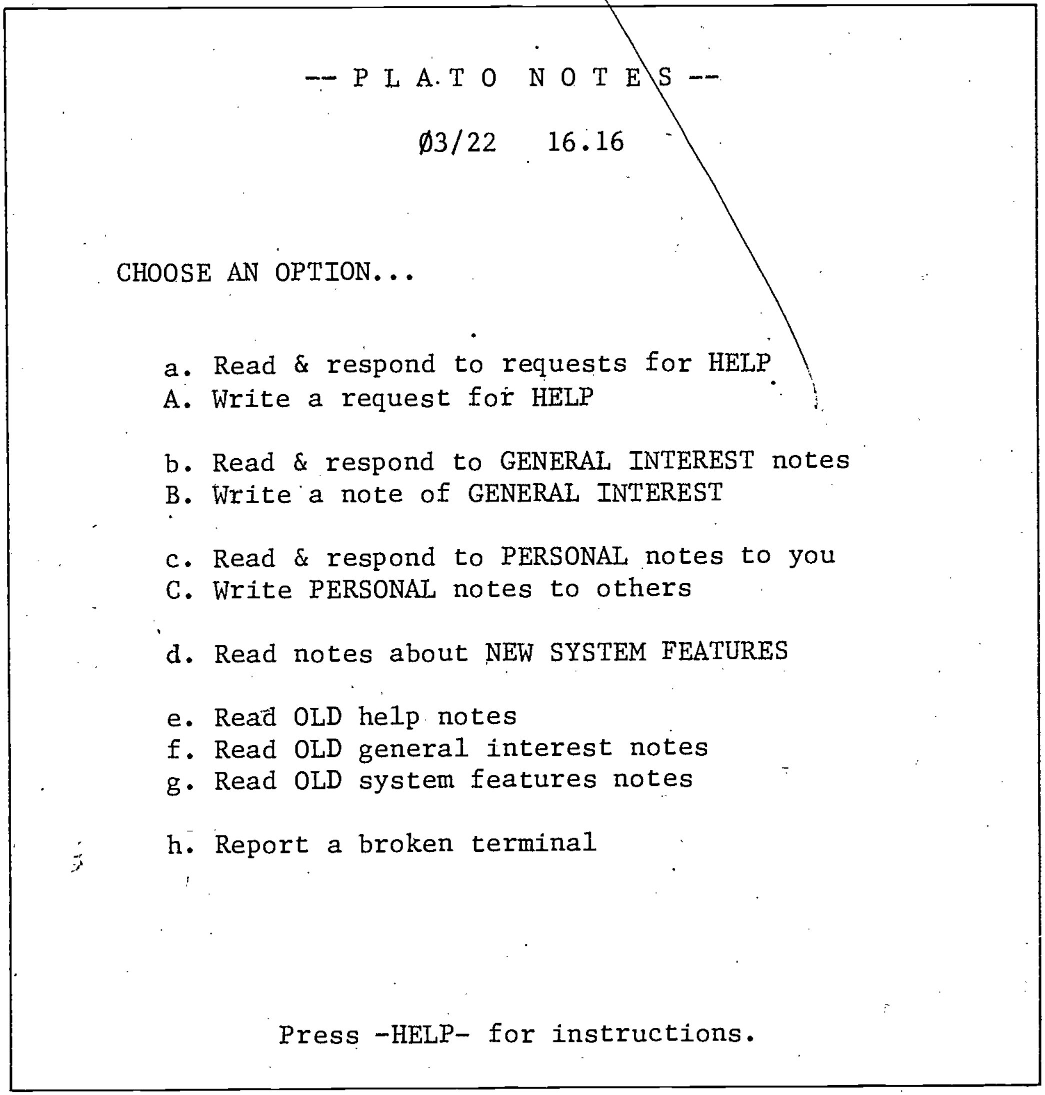
In the "New System Features" section, systems programmers enter brief descriptions of new TUTOR features or changes in existing TUTOR features. These "New System Features" are more thoroughly described in lesson "aids". The "General Interest Notes" section of lesson "notes" has entries from authors as well as systems programmers. Information on how to write notes and comments is available by pressing the HELP key while in "notes".

You should glance at both "New System Features" and "General Interest Notes" every day or two. All notes have identifying keywords so you can survey the topic list to decide which ones you want to read. If you do not read these two sections of "notes", you will not be aware of changes in TUTOR; and you will miss the opportunity to take part in the discussion of what directions PLATO and TUTOR should take.

Getting Help

There may be times when you need additional help for your specific problem. A TUTOR consultant can be contacted via the terminal by pressing the TERM key and entering "consult" (no quote marks). Personal notes may also be sent directly to a consultant or your specific questions may be discussed in the HELP portion of lesson "notes". More details on obtaining assistance are available in "aids".

A more direct way of getting help is to ask the person at the next terminal. If he is an author, he might be able to help you. Generally the attitude of authors is one of willingness to help other authors if they can. Most PLATO users have a spirit of cooperation!



```
                  -- P L A T O   N O T E S --

                        Ø3/22   16.16


   CHOOSE AN OPTION...


      a. Read & respond to requests for HELP
      A. Write a request for HELP

      b. Read & respond to GENERAL INTEREST notes
      B. Write a note of GENERAL INTEREST

      c. Read & respond to PERSONAL notes to you
      C. Write PERSONAL notes to others

      d. Read notes about NEW SYSTEM FEATURES

      e. Read OLD help notes
      f. Read OLD general interest notes
      g. Read OLD system features notes

      h. Report a broken terminal



                Press -HELP- for instructions.
```

Figure 13.1 Index display from lesson "notes"

On rare occasions you will want a print of your lesson to track down a program error. To obtain a print, press shift-DATA from the Author Mode display; then choose option "p".

Other Features and Resources

There are several TUTOR lessons which contain general information on courseware currently available. These lessons are summarized in figure 13.2.

Student Data -- It is possible to collect data on student performance in a lesson or parts of a lesson. Data such as the number of arrows the student encountered, the amount of time he spent in an area of the lesson, the number of correct and incorrect responses, and the number of times he requested help may be collected. Correct and incorrect student responses may also be collected and reviewed by the lesson author or instructor. This data is automatically stored by PLATO and is available in both a formative and summative manner. Lesson "aids" has a complete description of all data options.

Router Lessons -- There are TUTOR lessons which can be used to organize many instructional modules into a coherent set of modules. The set of instructional modules can be written by one author, a group of authors, or many different authors. The router lesson is used to access each module in the appropriate sequence for your students. There is a complete description in lesson "aids".

PEER Group -- The activities of the PLATO Educational Evaluation and Research Group are partially listed in figure 13.3. There is an on-line statistical package available to all authors in a PLATO lesson named "stat".

lesson name	use in student or author mode	description
sample	student mode	allows one to examine PLATO lessons in many subject areas
topics	student mode	catalog of PLATO materials which have been used with students
catalog	author mode	short description of all lessons on PLATO
authors	author mode	listing of all PLATO authors

Figure 13.2 Lessons containing general information on courseware

PEER GROUP -- Research and Service Program

Student interactions with PLATO
Evaluation as an approach to instructional design
Measurement and prediction of instructional effectiveness
Maintenance of CERL statistical package
Direct consultation with authors in evaluation, design, and
implementation

Figure 13.3 PEER group activities

Appendix A: Editing in TUTOR

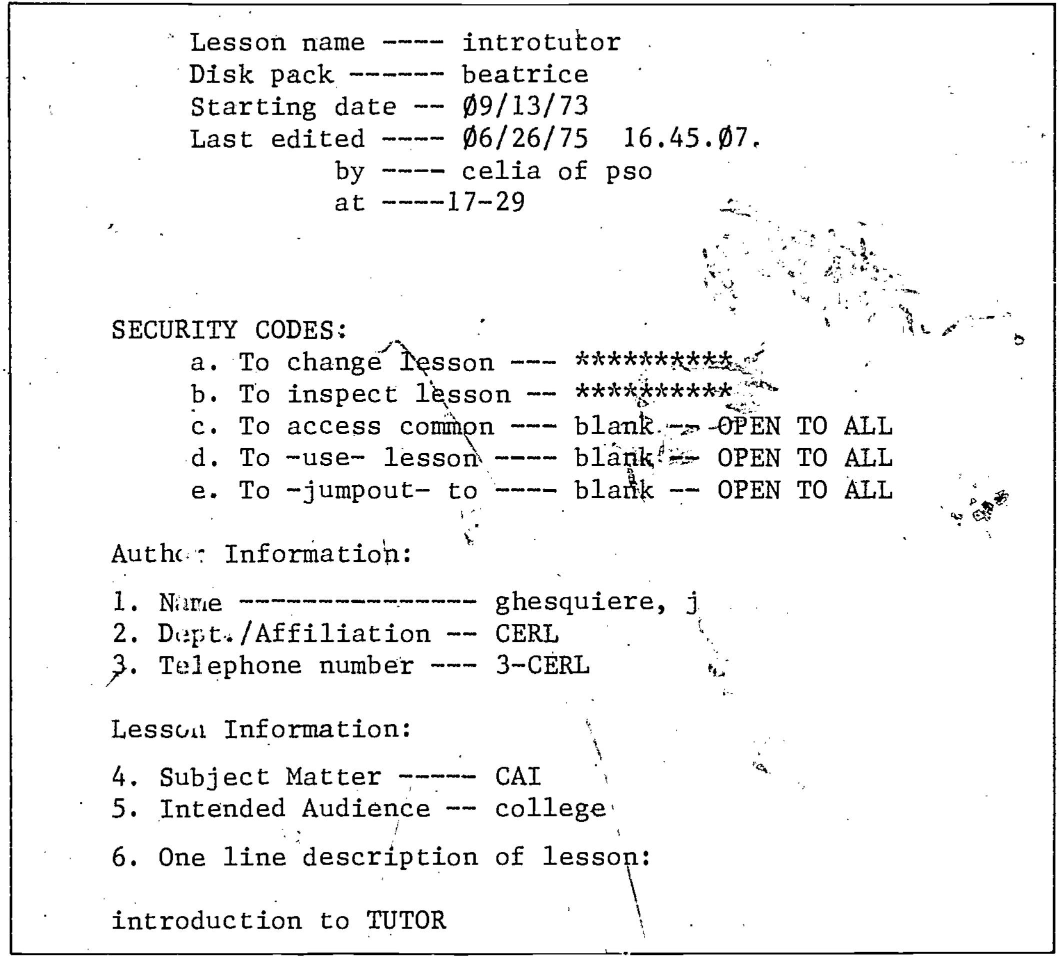
Lesson Data

Before learning to edit in TUTOR you should be familiar with the keyset. If you have not done so already, use lesson "help" as a student for an introduction to the keyset. You should also be familiar with the sign-on procedure as explained by your site dire ʼr.

When you are editing your lesson you are actually in a TUTOR lesson called the "editor". The TUTOR editor has many aids for authors incorporated into its structure. You may obtain aid by pressing the HELP key. Consequently there is little need for a comprehensive printed guide to TUTOR editing. This appendix is meant to introduce you to editing. After you know a few editing directives, you can use the HELP key in the editor to learn more advanced options which you will not formally learn now.

To start learning how to edit, enter your lesson name on the author mode display. The first time you enter your lesson, you should fill in the descriptive data requested on the DATA display, figure A.1. Type the number corresponding to the data you wish to enter, type the information requested, then press NEXT. 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 you 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 computer operator! If you leave the "change code" blank, anyone may edit your lesson, so enter some change code. The "inspect code" is the security code anyone must have to look at your lesson. If you do not mind who sees your lesson, 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 pressing DATA.

When you press NEXT from the lesson data display you will reach the block listing display which lists the name of your lesson and block names. If yours is a new lesson, it will have only one block, "a".

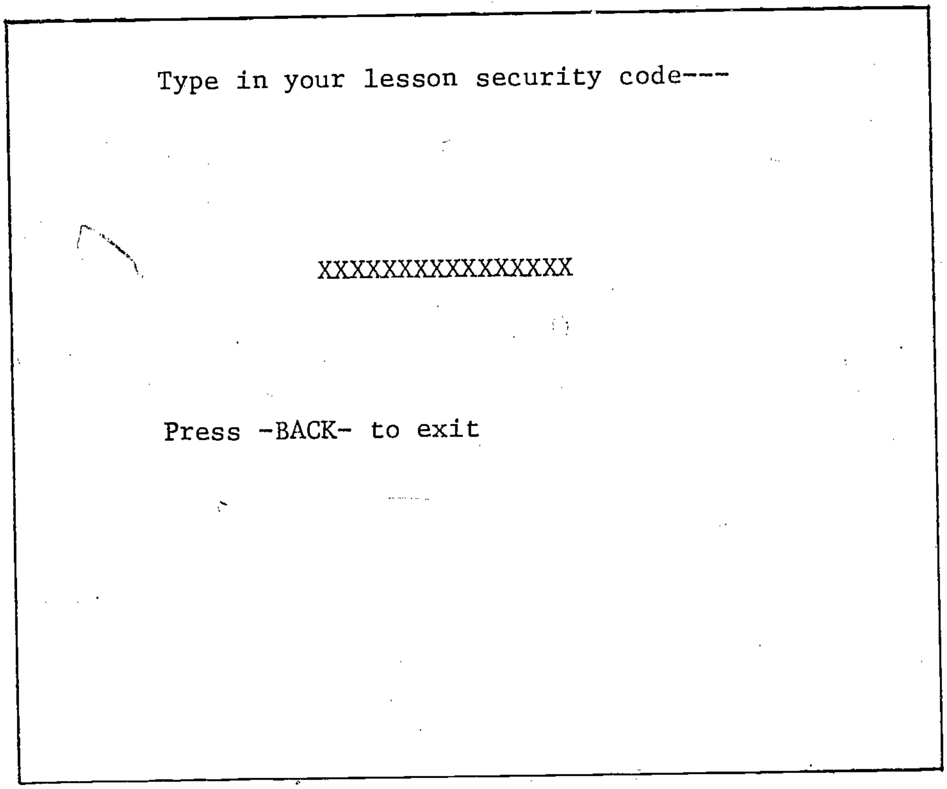


```
          Lesson name ---- introtutor
          Disk pack ------ beatrice
          Starting date -- Ø9/13/73
          Last edited ---- Ø6/26/75  16.45.Ø7.
                   by ---- celia of pso
                   at ----17-29


     SECURITY CODES:
          a. To change lesson --- **********
          b. To inspect lesson -- **********
          c. To access common --- blank -- OPEN TO ALL
          d. To -use- lesson ---- blank -- OPEN TO ALL
          e. To -jumpout- to ---- blank -- OPEN TO ALL

     Author Information:

     1. Name --------------- ghesquiere, j
     2. Dept./Affiliation -- CERL
     3. Telephone number --- 3-CERL

     Lesson Information:

     4. Subject Matter ----- CAI
     5. Intended Audience -- college

     6. One line description of lesson:

     introduction to TUTOR
```

Figure A.1 Lesson data display

Lesson Security Code (Lesson Change Code)

On the lesson data display you entered a change code for your lesson. But, as of now, PLATO has no change code associated with your records. It only knows a name, a course, and your sign-on password. You will need to enter a security code which PLATO stores with your records to allow you to edit your lesson. To do this, return to the Author Mode display by pressing BACK; then press shift-DATA. Choose option "s", which should bring you to a display resembling figure A.2. At this "lesson security code" display type the same code you entered as a change code for your lesson. A random number of XXX's will appear as you type to prevent anyone from seeing the letters you are typing. Then press NEXT to return to the Author Mode page. Now reenter your lesson by typing the name and pressing NEXT.



Type in your lesson security code---

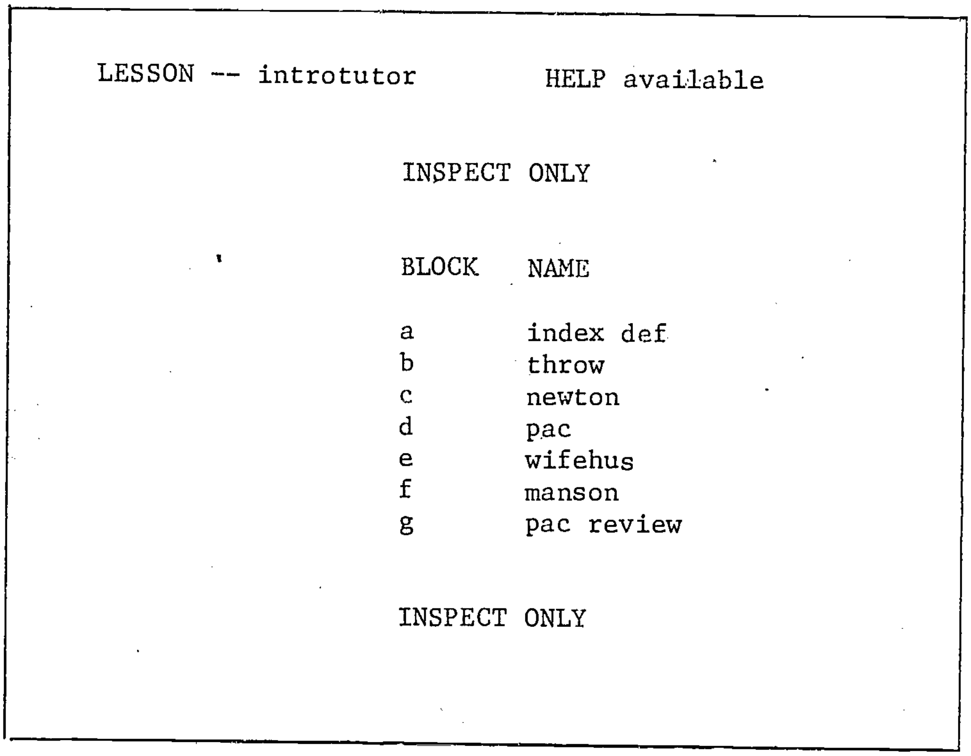
XXXXXXXXXXXXXXXX

Press -BACK- to exit

Figure A.2 Lesson security code display

INSPECT ONLY as an Author

If the lesson security code stored with your records does NOT match the lesson security code entered on the lesson data display, figure A.1, then you can only inspect a TUTOR lesson in author mode. If your block listing display has the "INSPECT ONLY" message, as in figure A.3, you cannot change any code in that lesson. That is, you cannot insert, delete, or replace any TUTOR code. If your block listing display has the "INSPECT ONLY" message, return to the "Lesson Security Code" section, page A.2.



```
LESSON -- introtutor        HELP available

               INSPECT ONLY

               BLOCK   NAME

               a       index def
               b       throw
               c       newton
               d       pac
               e       wifehus
               f       manson
               g       pac review

               INSPECT ONLY
```

Figure A.3 Block Listing Display, INSPECT ONLY

Inserting TUTOR Code

From the block listing display enter block "a" by pressing "a". You are now on the "line display". To begin inserting TUTOR code, type the letter "i", which stands for "insert", and press NEXT. Now you are ready to type a line of TUTOR code. You may wish to use unit "gorge" from Chapter 1 for practice in editing. It is reprinted here for convenience.

```
unit    gorge
at      12Ø3
write   Where is Louis S. B. Leaky's anthropological dig?
arrow   14Ø1
specs   bumpshift,okspell
answer  <the,it,is,in,at,kenya> olduvai (gorge,canyon)
write   Homo habilis was discovered there.
wrong   <it,is,the,at,in,tanzania> gombe stream research cen
        ter
write   That's the site of Jane Goodall's work with chimpanz
        ees.
```

Type the command -unit-. Then press TAB so the tag portion of the line is spaced over, and type the tag "gorge". (Do not type the hyphens or quote marks in your lesson.) Press NEXT to enter the second line of TUTOR code. When you have finished inserting the second line, press BACK to return to the line display.

To insert code someplace after the top line of the display, type "i" and then the number of the line after which you want to insert, then press NEXT. Type "i2" now, and press NEXT to insert the third line of unit "gorge".

When you enter a block you are always at the first, or top, line of the block To go forward, type "f", then the number of lines you want to roll the display forward, then press NEXT. To go backward, or down, type "b", followed by the number of lines, then press NEXT. With "i", "f", and "b" as well as many other editing directives, if you do not enter a number, PLATO assumes a 1. Thus "f" is equivalent to "f1". Practice using these three directives now. Insert the rest of unit "gorge" and move forward and back until you feel comfortable using these directives.

Deleting and Replacing Lines

To delete a line, you must first bring the line to the top of the display. When the offending line is at the top, type "d", then press shift-HELP to delete it. If you have a superfluous line in your unit "gorge", bring it to the top and delete it now. (If your unit "gorge" is correct, choose some line to delete for practice. You can then insert it again.) There is one caution to observe when using "d": "d1Ø", for instance, means "delete 1Ø lines starting with the top line." It does not mean "delete the tenth line'. When you delete, you must bring the unwanted line to the top of the display before deleting.

To replace a line, type "r" and then the line number (if no number is entered, a 1 is again assumed) then press NEXT. Retype the line. Then press either NEXT to replace the following line, or BACK to stop replacing. Practice using "r" now. Replace a line containing typing errors, if you have one; if not, replace any line for practice.

The COPY and EDIT keys

The COPY key (which may have a carat on the keycap, Λ) will copy into a line being "inserted" or "replaced" from the line appearing immediately above it on the panel. Each press of the COPY key will copy another "word" (any characters bounded by spaces or punctuation) into the line being worked on. At any point, pressing the shift-COPY key will copy the rest of the line.

The first press of the EDIT key (to the right of the NEXT key) will erase the entire line being worked on. Successive presses of EDIT will return "words" to that line, one at a time, in a fashion similar to the COPY key. These keys are useful for correcting errors without having to retype an entire line. Practice using COPY and EDIT while inserting or replacing until you are familiar with their use.

Testing Your Lesson in Student Mode

From either the line display or the block display, you can press shift-STOP to enter your lesson in student mode. Do this now. If there are no errors in your TUTOR code, you will see the unit as it appears to a student. If your code has errors, you will see a display entitled "condense errors and warnings". On this display is listed TUTOR code which PLATO could not interpret, the name of the unit in which the error occurs, and the lesson part and block. ("Lesson part" refers to the number of block display pages, or "parts", in a lesson. Almost all TUTOR lessons are 1-part lessons.) Some common condense errors include mistyping a TUTOR command or forgetting to press TAB between the command and the tag. You should correct the errors before using the lesson in student mode. Press shift-STOP if you had any condense errors. This returns you to the block display. If you press NEXT from the condense errors display, PLATO will attempt to execute your lesson ignoring the lines containing errors. After you have corrected the errors, condense the lesson again (shift-STOP). To return from student mode to the author options display, press shift-STOP.

Additional Editing Options

Complete instructions are always available when you are editing. Press HELP from almost any display for a description of the options available while at that display.

To create a new block, press the shifted letter of the block you want to add after. For example, press shift-a to create a block after block "a". After creating a block you must insert something in it, or it will disappear. An exception to this is block "a"; it is always there, whether it contains anything or not.

To move from one block to another, first return to the block display by pressing BACK, then enter the other block by typing its letter.

Figure A.4 summarizes the various displays of author mode and gives the keys required to get from one display to another. You may wish to press HELP from each display and glance briefly at the options and directives available. Do not try to memorize them; you will find it easier to learn them as you need them.

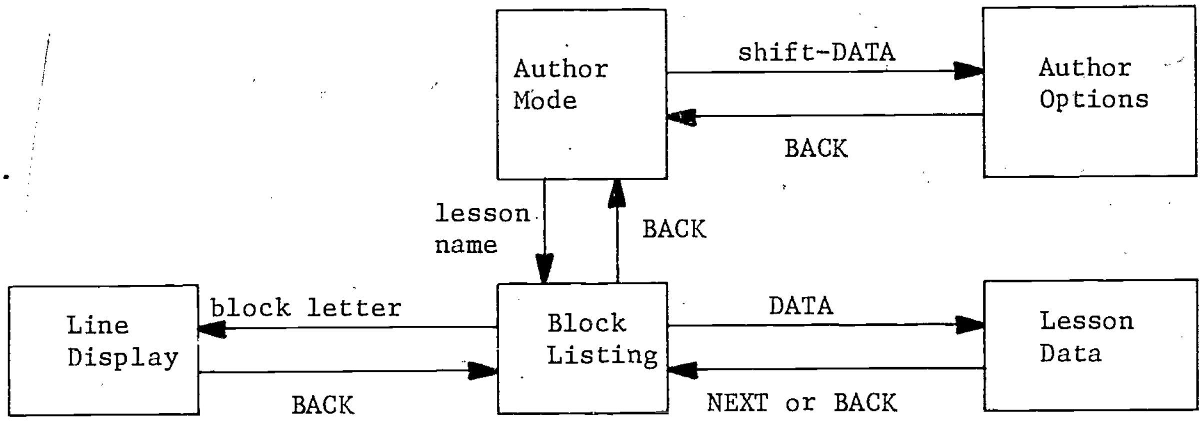


Figure A.4 Flow diagram of editing displays

Appendix B: Summary of TUTOR Commands, -specs- Options, and Systems Reserved Words

This list contains only those commands, -specs- options, and systems reserved words described <u>in this book</u>. For others, see lesson "aids".

TUTOR Commands:

*	ansv	specifies numerical answer, a tolerance (percent or absolute) is allowed
*	answer	specifies a correct student response
	arrow	plots arrow on panel at tag location and allows student input
	at	specifies a location on the panel (either coarse or fine grid)
	back	specifies unit to proceed to if BACK key is pressed
	back1	specifies unit to proceed to if BACK1 is pressed
	base	alters base unit to unit named in tag or clears base pointer if no tag is used
	calc	stores a value in a variable and permits general computational operations
	calcc	conditionally performs one of a list of calculations, depending on value of an expression
	calcs	sets a variable to one of a list of values depending on value of an expression
	charset	loads specified character set into terminal
	circle	draws a circle or arc on the panel; tag (fine grid) specifies radius and center
	circleb	draws broken circle or arc
*	concept	specifies idea of a correct student response; also see -vocabs-
	data	specifies unit to proceed to if DATA key is pressed
	data1	specifies unit to proceed to if DATA1 is pressed
	define	allows author to give names to student variables. A defined name must be 7 or fewer characters and must not begin with a number or operator
	do	attaches unit named in tag of -do- to unit in which -do- command appears
	draw	draws lines between panel locations given in tag
	end	ends a help sequence (tag "help") or a lesson (tag "lesson")

* indicates judging command, all non-starred commands are regular

endarrow delimits commands in a unit pertinent to the preceeding arrow

erase erases number of characters (1 argument tag) or block of characters and lines (2 argument tag) or entire panel (no tag)

* exact specifies correct student response, including punctuation and spaces

* exactc conditional form of -exact- command

finish specifies which unit will be executed at exit via STOP1

goto attaches unit named in tag to present unit, but commands below -goto- are not executed

help specifies unit to proceed to if HELP key is pressed

help1 specifies unit to proceed to if HELP1 is pressed

inhibit prevents normal action of TUTOR feature described in tag.
Tag "erase" -- display is not erased when main unit is changed.
Tag "arrow" -- arrow is not displayed on panel.

* join attaches unit named in tag to unit in which -join- command appears (judging <u>and</u> regular)

judge alters previous judgment; -judge- is a regular command

jump specifies unit to branch to immediately

lab specifies unit to proceed to if LAB key is pressed

lab1 specifies unit to proceed to if LAB1 is pressed

* match searches student response for occurrence of words listed in tag, sets a variable and judges response

micro specifies micro table

mode changes terminal from present mode to mode named in tag (write, erase, rewrite)

modperm copies modperm list into setperm list

next specifies unit to proceed to when NEXT key is pressed

next1 specifies unit to proceed to if NEXT1 is pressed

* no judges any response "no"

* ok judges any response "ok"

pause causes PLATO to wait before continuing execution of unit

randp chooses a random number from a field specified by -setperm-, and stores it in variable named in tag. The -randp- command causes sampling without replacement.

randu 2 argument tag: chooses an integer between 1 and the integer in the second argument of the tag, and stores it in the variable named in the first argument.
1 argument tag: chooses a fractional number between Ø.Ø and 1.Ø and stores it in the variable named in the tag. The -randu- command causes sampling with replacement.

* indicates judging command, all non-starred commands are regular

rdraw draws figure which can be sized and rotated

remove removes a number from the modperm list

restart specifies the lesson and unit in which the student will be when he returns to PLATO

rotate rotates sized writing. Tag is the number of degrees the writing is to be rotated counter clockwise from the horizontal axis.

setperm sets up a number field from 1 to the number given in tag

show displays value of variable or expression named in tag

showa displays alphanumeric characters stored in variable named in tag

showe displays value of variable or expression in exponential format

showo displays value of variable or expression in octal (base 8) notation

showt displays value of variable or expression in a format suitable for tables

size specifies size of writing. Size Ø is normal writing.

slide projects a slide on the panel

*specs specifies judging options, and acts as a marker to be returned to when judging state is complete

*store calculates numerical value of student response and stores it in variable named in tag

*storea stores alphanumeric student response in variable named in tag

*storen removes a numerical portion of a student response and stores the result in variable named in tag

term allows branch to unit containing the -term- command when the student presses TERM and types word which matches tag of -term- command

unit identifies a section of a lesson; a unit name may be up to 8 characters long

vocabs specifies a vocabulary list for subsequent -concept- command

write puts text on the panel

writec conditional form of -write- command

*wrong specifies an incorrect student response

*wrongv specifies an incorrect numerical response, a tolerance (percent or absolute) is allowed

zero stores the value Ø in the variable named in the tag. The 2 argument form zeros a block of consecutive variables beginning at the variable named in the first argument and continuing for the number of variables in the second argument.

* and $$ allow author comments

* indicates judging command, all non-starred commands are regular

-specs- Options:

bumpshift	capitals in the student's response are ignored
nookno	"ok" or "no" are not written after judgment
noops	mathematical operators are not allowed in student's response
okextra	extra words in student's response are ignored
okspell	student's response is judged "ok" even if it contains spelling errors

Systems Reserved Words:

jcount	number of characters in student's response
judged	equals -1 if student's response was judged "ok", Ø if response matched -wrong- or -wrongv- tag, 1 if no match was made or a -no- was encountered
mode	equals -1 if in erase mode, Ø if in rewrite mode, 1 if in write mode
ntries	number of attempts student has made at this arrow
opcnt	number of mathematical operators in student's response
where	coarse grid location of last panel activity
wherex	fine grid x location of last panel activity
wherey	fine grid y location of last panel activity

Index

Checklist for using "Introduction to TUTOR" and "introtutor"*

your checklist	items to be read and exercises to be completed
	read chapter 1; pp 1.1 to 1.9
	read Appendix A and practice the editing directives
	read section 1 of "introtutor"
	do exercise A; pp 1.10 to 1.18 (refer to sample lesson A in "introtutor")
	read chapter 2; pp 2.1 to 2.14
	read section 2 of "introtutor"
	do exercise B; pp 2.15 to 2.21
	read chapter 3; pp 3.1 to 3.9
	read section 3 of "introtutor"
	do exercise C; pp 3.10 to 3.13
	read chapter 4; pp 4.1 to 4.3
	read section 4 of "introtutor"
	do exercise D; pp 4.3 to 4.9
	read chapter 5; pp 5.1 to 5.7
	read section 5 of "introtutor"
	do exercise E; pp 5.8 to 5.21
	read chapter 6; pp 6.1 to 6.8
	read section 6 of "introtutor"
	do exercise F; pp 6.8 to 6.14
	read chapter 7; pp 7.1 to 7.7
	read section 7 of "introtutor"
	read the remaining chapters of "Introduction to TUTOR"
	read section 8 of "introtutor"

* "introtutor" is a TUTOR lesson used in student mode.