DOCUMENT RESUME

ED 050 583	EM 008 943
AUTHOR	Avner, R. A.; Tenczar, Paul
TITLE	The TUTOR Manual.
INSTITUTION	Illinois Univ., Urbana. Computer-Based Education Lab.
SPONS AGENCY	Office of Education (DHEW), Washington, D.C.
REPORT NO	R-CERL-X-4
PUB DATE	Mar 70
CONTRACT	OEC-6-10-184
NOTE	194p.; See also User's Memos, EM 008 944, and EM 008 945
EDRS PRICE	EDRS Price MF-$0.65 HC-$6.58
DESCRIPTORS	*Computer Assisted Instruction, Computer Graphics, Computer Programs, *Curriculum Development, Display Systems, *Manuals, *Programing Languages
IDENTIFIERS	PLATO, Programmed Logic for Automated Teaching Operations, *TUTOR

ABSTRACT

The TUTOR logic-building language to be used with the PLATO (Programmed Logic for Automated Teaching Operations) system is explained in this manual. TUTOR is designed to transcend the difficulties of FORTRAN for a computer-based educational system utilizing graphical screen displays. The language consists of about seventy words or "commands" which can be used in various combinations to produce the desired effect. It was designed specifically for use by lesson authors lacking prior knowledge of and experience with computers. Although authors are able to write parts of useful lessons after approximately one hour of introduction to TUTOR, the ultimate complexity and flexibility of TUTOR lessons is limited largely by the ingenuity and experience of lesson authors. A sample TUTOR program which allows the student to construct geometric shapes on his television screen demonstrates how a TUTOR lesson appears to the student. A complete description of the structure and elements of the language is presented, as well as a description of methods for inputting lessons and obtaining output. The manual is intended to be used as a textbook by the beginning lesson author and as a reference tool by the experienced TUTOR user. (JY)

CERL REPORT X-4

JANUARY, 1969

REPRINTED MARCH, 1970

THE TUTOR MANUAL

R.A. AVNER
PAUL TENCZAR

Computer-based Education Research Laboratory

University of Illinois Urbana Illinois

1

This work was supported in part by the National Science Foundation under Contract NSF GJ-81, and in part by the Advanced Research Projects Agency through the Office of Naval Research under Contract Nonr 3985(08).

THE TUTOR MANUAL

R. A. Avner
Paul Tenczar

Computer-based Education Research Laboratory
University of Illinois - Urbana
1969

ACKNOWLEDGEMENTS

MANUAL

We would like to thank Richard W. Blomme and William Golden for the time and effort they spent reading drafts of this manual. Any awkward grammar or confusing concepts in this manual are only a drop remaining from the ocean of errors eliminated by them.

Each of the co-authors is willing to assign responsibility for all remaining errors to the other co-author.

R. A. Avner
Paul Tenczar
January, 1969

TUTOR LOGIC

TUTOR was conceived in June, 1967 because of my desire for a simple users language transcending the difficulties of FORTRAN and designed specifically for a computer-based educational system utilizing graphical screen displays. Since this is the first published account of TUTOR, I would like to mention the many people who have helped me in TUTOR's development. Richard Blomme must be singled-out for his ideas and programming which have encompassed all aspects of TUTOR. Indeed, since September, 1968, he has taken over many of the responsibilities for the continued development of TUTOR. At that time, I resumed work on my Ph.D. in Zoology, and with R. A. Avner, began writing The TUTOR Manual. Other persons who added ideas and programming effort to TUTOR were R.A. Avner, Robert Bohn, John Gilpin, J. Richard Dennis, William Golden, Robert Grandy, Don Lund, Phillip Mast, James Payne, and Louis Steinberg.

Paul Tenczar
January, 1969

FINANCIAL SUPPORT

Production of this manual was supported in part by the Advanced Research Projects Agency through the Office of Naval Research under Contract Nonr-3985(08), by the National Science Foundation under NSF GJ-81, and by the U.S. Office of Education under OE-6-10-184.

Table of Contents

(continued on next page)

Chapter 1

INTRODUCTION

The PLATO Computer-Based Education System was designed to aid both student and instructor in the educational process through use of the capabilities of the modern digital computer. The PLATO computer interacts with each student by presenting information and reacting to student responses. The computer's actions follow the instructor's rules which specify what is to be done in each and every possible situation. A lesson constructed of such a set of rules can have a flexibility approaching that possible when each student has a human tutor. In fact, the rules defining a useful tutorial lesson presented by computer are quite similar to those implicitly used by a human teacher. For example, areas in which a student has proven competence are given minimal coverage while areas in which the student lacks competence are developed more thoroughly.

In constructing computer lessons, instructors must use "languages" w[illegible] allow communication with computers. One such language is TUTOR.

TUTOR consists of about seventy words or "commands" which can be used in various combinations to produce desired effects. Much lesson writing can be done using less than a dozen of these commands. TUTOR was designed by Paul Tenczar of the Computer-Based Education Research Laboratory specifically for use by lesson authors lacking prior experience with computers. The language is extremely easy to learn and to use. Normally, authors are able to write parts of useful lessons after a one-hour introduction to TUTOR. The simplicity of TUTOR does not limit its applications. Since TUTOR is a true language, the ultimate complexity and flexibility of TUTOR lessons is limited largely by the ingenuity and experience of lesson authors.

TUTOR is among the languages presently used on the PLATO system at the University of Illinois. PLATO (an acronym for Programmed Logic for Automated Teaching Operations) currently consists of a CDC 1604 computer, student stations, and the equipment necessary for the computer to interact with these stations. Each student station has a television screen for presentation of information to the student and a keyboard used for student communication with the computer. Chapter 2 will describe a typical student station and show how PLATO communicates with a student through such a station.

SOME REASONS FOR USING TUTOR AND THE PLATO SYSTEM

1. Individual Attention

In contrast to a conventional classroom in which a teacher manages twenty to thirty students simultaneously and can seldom give special attention to individual students, PLATO appears to give each student undivided attention. This appearance results from the computer's ability to identify and handle most student requests in a small fraction of a second. When several students request material simultaneously, the PLATO system processes their requests in turn. However, PLATO works so rapidly that the last processed student seldom has to wait more than one-tenth of a second for a reply from the computer. To most students, one-tenth of a second appears to be instantaneous. One aspect of individual attention is rapid feedback. The student can get immediate knowledge of the correctness of his responses.

PLATO's individual attention capability together with its computational and graphic display abilities allows authors to produce simulated laboratories in which each student can collect his own data without fear of damage to himself or apparatus. Since the time scale of a model laboratory can be shortened, the student does not have to wait hours, days, or even years for actual experimental conditions to occur. In one University of Illinois course the student is allowed to experiment with a model home thermostat system. The student selects the outside daily temperature range, the furnace and air conditioner thermostat settings, the type of furnace and air conditioner, and the type of house insulation. The student can then see a graph of indoor and outdoor temperatures for a twenty-four

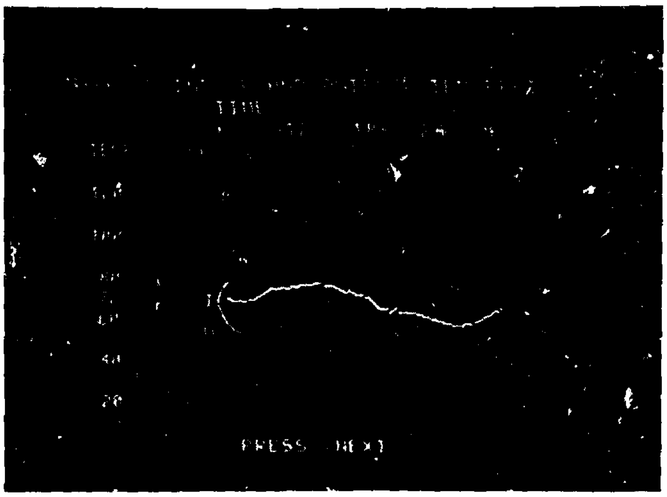


Figure 1.1

hour period under these conditions. (see Figure 1.1) Another model laboratory allows students to train a simulated mouse in an operant learning situation. The "mouse" moves about the screen of the student's TV set in response to "stimuli" given by the student and past "experience."

At another level, TUTOR permits the author to provide alternate information based on a pattern or history of student response. Unlike many forms of programmed instruction, computer-based education is not limited to providing lesson alternatives based on one student response. Thus, in a TUTOR lesson it would be possible to give a student remedial instruction if he missed, say, any four of the last ten questions. Some of the techniques of lesson individualization are discussed in Chapter 4.

2. Ease in Lesson Construction

Additions or corrections to TUTOR lessons can be made faster and easier than additions or corrections to a typewritten manuscript. A single change will affect all students using the lesson. Thus, if an author finds that recent advances in his field have outdated a section in his lesson, he need only sit down at any PLATO station and replace the outdated section with the latest information. The updated lesson is immediately available for student use. With languages such as TUTOR, the author spends most of his time working with lesson content rather than struggling to "interpret" the content to the educational medium being used.

3. Complete Data Handling

Authors, especially at early stages of lesson development, can collect data on all student responses. If students are having difficulty with certain concepts (as shown by the number of incorrect responses or amount of time spent in the area), the author can use the data to alter the lesson and clarify the difficult areas. Automatic data collection is also a useful tool for experimental studies in the behavioral sciences. The computer acts as a very accurate and unbiased data collector for the experimenter. Data such as response times and answer scores can also be used during a student lesson as criteria for choosing the next lesson segment for the student.

4. Computational Ability

In addition to recording student data, authors in experimental studies can use the computer to perform necessary statistical operations on data as they are being collected. The computational ability can also serve the student in subject areas requiring the student to perform lengthy or complex mathematical operations. Freed from tedious calculations, the student can rapidly explore the important relationships among elements of a problem.

5. Visual Displays

PLATO can be used to select and present stored material such as printed messages or photographic slides. In addition, PLATO is also able to construct geometric figures or graphs. Such constructed displays are produced by the computer following instructions specified by either the author or the student. A constructed graphic display might, for example, be used to allow a student in a Physics course to specify the shape and composition of a lens. PLATO could then produce a side view of the lens on the student's TV screen. Upon the student's request, PLATO might also show the path of light rays through this model lens. Chapter 2 illustrates a TUTOR program which allows the student to construct geometric figures on his TV screen. This same program can also evaluate the student's work.

6. Judging

The TUTOR language allows the author to specify a wide range of criteria for acceptable and unacceptable student responses. At the most limited level, the computer may require that the student respond in exactly one way (e.g., the answer "4") or require that the student respond with one of a list of correct answers (e.g., "4", "4.0," "FOUR"). TUTOR goes beyond this restricted form of "answer matching." The student's answer can consist of single words, a phrase, and even sentences. The computer can be directed to indicate to the student such things as possible misspellings, incomplete answers, duplicate terms in lists, or incorrect words in sentences. An author need not specify every possible form of correct answer. In certain instances the author might even let PLATO decide what the correct answer is. PLATO's decision would be based on rules given by the lesson author. For example, the student might be allowed to construct his own addition problems. The "correct answer" would be determined by the rule "sum the factors given by the student." Chapters 3 and 5 will cover further examples of judging options available to the TUTOR author.

7. Drill and Practice

With little effort an instructor using TUTOR can provide his students with an untiring drill master. The author can set the rules of timing, problem removal, and criteria for exercise completion or he can allow the student to set his own rules. For example, a University of Illinois French course allows the student to choose his timing for an English-to-French translation drill. Words from a list are presented in random order. A word correctly translated within the time limit is removed from the list. Thus, near the end of the drill the list contains the words most difficult for the student. When finished, the student can take the drill again at a quicker pace. After the drill session or before the next lesson session, PLATO can remind the student which items in the drill caused the most difficulty. The accounting ability of the computer is ideally suited to take over this tedious and time-consuming job from teachers and release them for more rewarding labors. In addition, a well written computer drill lesson can hold student attention through its game-like qualities.

ON THE MISUSE OF COMPUTER-BASED EDUCATION

Balancing the positive features of PLATO are several negative considerations. These considerations can be placed in two general classifications; cost and relative effectiveness. If instructional material can be presented with equal effectiveness by any of several media, that medium which has the lowest cost in time and/or money is usually chosen. Even when one medium is superior to another there are usually financial limitations. Cost and effectiveness are intertwined in any actual judgment but their separation simplifies further discussion.

Cost. In future versions of PLATO the cost per student-contact-hour will approach that incurred in conventional elementary school education. This favorable cost relationship does not presently exist for any computer-based educational system. The effect of the present high purchase and operating costs limits the availability of computer-based educational systems. In addition, existing systems are generally subject to heavy use which limits new research and lesson development.

Of more direct interest to the author is the cost in effort needed to prepare lesson material. Certain instructional techniques, such as programmed instruction or inquiry learning, are costly in preparation time, whatever the medium of presentation. Other techniques, such as drill-type lessons, are relatively easy to prepare. In addition, cost is a function of the complexity of the materials. It is easier to give a thorough coverage of the rules of integer addition than to do equal justice to, say, the laws of thermodynamics or irregular French verbs. The use of languages such as TUTOR minimizes the effort needed to put draft lessons onto the computer. Thus, the preparation of lesson content will usually be the most time-consuming aspect of preparing lessons for PLATO. The instructor must judge in each case whether the preparation cost will be balanced by the benefits to students.

Relative effectiveness. The instructor is to some extent an expert in teaching his material. He generally has at least a notion of an ideal method or methods by which this material could be presented to maximize learning. Computer-based education should be considered as merely another medium which might allow some of these methods to be used in practice. *The instructor will seldom go wrong if he lets the message dictate the medium.* Many of the trivial or contrived uses of computers and other "glamorous" media are the result of instructors who start with a commitment to a medium and proceed to write materials intended to utilize the features of that medium. No magical improvement of material occurs simply because of presentation by a computer-based educational system.

Instructors using both cost and effectiveness as guides can avoid the mistake of using computer-based education in situations where more effective media exist. The very flexibility of systems such as PLATO tends to draw lesson authors into the trap of acting as if everything that *can* be done by computer-based education *should* be.

HOW TO USE THIS MANUAL

This manual is intended for use both as a textbook for the beginning lesson author and as a reference manual for the experienced user of TUTOR. The manual is written for the prospective lesson author who has neither a background nor a particular interest in computers. Examples are used whenever possible to clarify use of TUTOR and to provide models for simple applications of the language. Chapters 3 through 8 are concerned entirely with the elements and structure of a TUTOR lesson. Upon finishing chapter 3, one should be able to write simple lesson material. Chapter 9 tells how this material can be put onto the computer.

Many parts of the manual on first reading will not appear to be of immediate use. In these cases it is generally quite sufficient to merely read for an understanding of *what* is possible rather than *how* it is done. Later, when a need for one of these techniques arises, one can return for a more thorough reading. Remember that useful lessons can be written while using only a fraction of the features of TUTOR.

The yellow-page section of the manual will be of most use to authors after they begin extensive lesson writing. The section contains a complete description plus useful examples of each of the available TUTOR commands.

Chapter 2

WHAT THE STUDENT SEES

Conversation between a student and his computer teacher occurs at a student station (see figure 2.1). The station is equipped with a television screen for display of the computer's part of the dialogue and a keyboard for the student to use in responding to the computer display.

Writing Keys

The keyboard (see figure 2.2) contains a set of keys labeled with alphabetic and numeric characters similar to those found on a typewriter. These keys are used by the student to answer questions. The computer "writes" what the student types on the screen in an appropriate place. Normally, pushing an alphabetic key causes a lower case letter to appear on the screen. If the shift key is held while a letter is typed, an upper case letter appears. However, lesson authors may choose to use only an upper case character set in which case unshifted keys produce upper case letters. Pressing the SUP key causes the next character to appear as a superscript while the SUB key produces a following subscript. Other keys have the effect of causing a carriage return and a backspace. The backspace can be used to superimpose characters.

The character set used by TUTOR is sufficient to write in any of the major European languages. In addition, keys exist which automatically superimpose an accent, grave, underline, overline, etc., over a preceding character. The computer automatically distinguishes between lower and upper case preceding characters so that the additional mark is positioned properly. Under directions of the lesson author, students can write using a Cyrillic or a Phonetic character set. Characters specific to a certain field of study (e.g., the sigma used so frequently in statistics) can be designed and employed in a lesson.

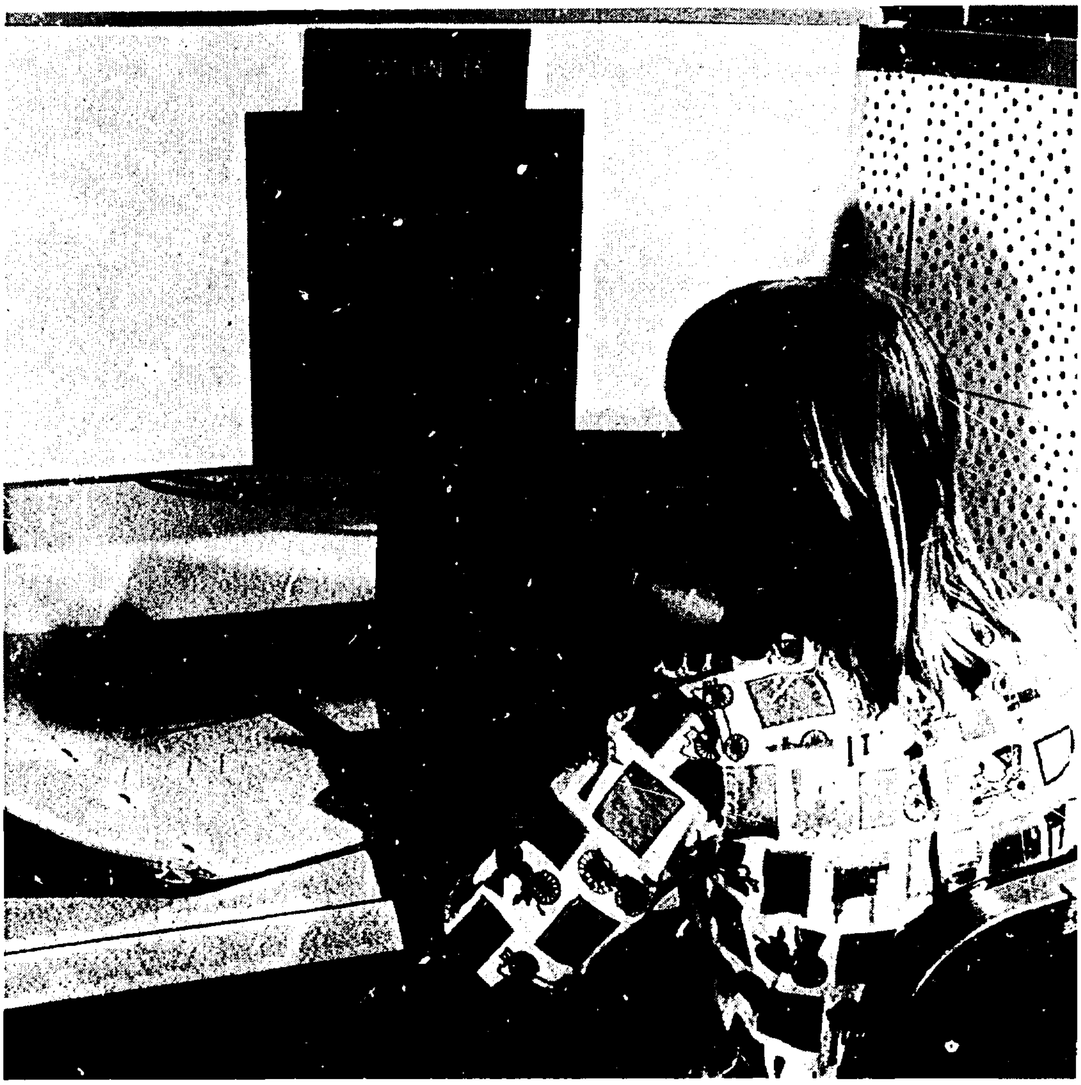


Figure 2.1

A student station

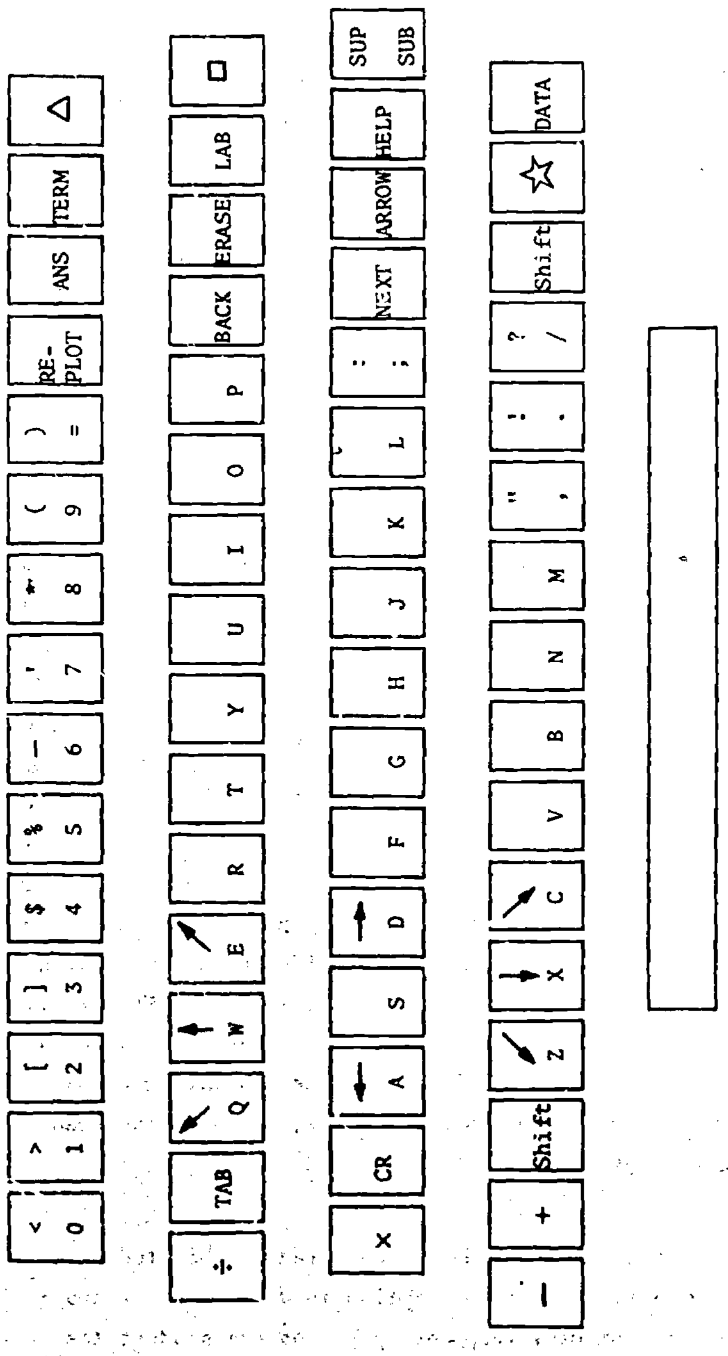


Figure 2.2

The Keyboard

Functional Keys

The keyboard also contains a set of keys labeled with words which represent lesson control options available to the student. A list of the main lesson control keys along with a description of their use follows.

NEXT

A student presses the NEXT key to request the computer to do the next logical action for the current student situation. For example, if the student has typed in an answer to a question he would press the NEXT key to obtain an evaluation of his answer. If the answer is incorrect, the student can push the NEXT key again which will in this case erase the student's response so that another answer can be typed in. If the answer is correct, pushing the NEXT key advances the student to the next question or informational display.

ERASE

A typing error can be deleted by pressing the ERASE key. The lower case ERASE key deletes only the last character typed while the upper case ERASE key deletes the entire response previously typed.

BACK

A student requests to review material previously seen by pressing the BACK key. Lesson authors provide the computer with information necessary to handle this student request. For example, a lesson author can allow the student to review previous material in reverse order, or the author can allow the student to choose which parts of the lesson he wants to review, or the author can disallow the BACK option.

ANS

The answer to a question may be requested by the student by pressing the ANS key. Unless this option is disallowed by the lesson author, the computer responds by giving the student the author's first answer choice for the question.

HELP

A student experiencing difficulty with the lesson can request aid by pushing the HELP key. The computer then takes the student into a sub-lesson segment provided by the author which is as specific to the student situation as the author desires. Upon completion of this supplementary lesson segment, the student is returned to the point in the lesson from which he requested aid. The additional lesson control option keys LAB and DATA function in a manner identical to the HELP key in branching the student to an author provided sub-lesson segment. These keys may be used to provide the student access to reference material.

TERM

A student can request the definition of any word by pressing the TERM key. The computer responds to this key by displaying the message "WHAT TERM?" near the bottom of the T.V. screen. The student can now type in the word he desires information about. He then presses the NEXT key. If the lesson author has provided for this word, the student obtains a sub-lesson segment of the author's choosing. Upon completion, the student is returned to the point in the lesson from which he pressed the TERM key. While these sub-lesson segments reached by a student through pushing the TERM key often concern word definitions, they can contain anything the author chooses. Thus, for a chemistry course the TERM key can provide access to information concerning the chemical elements similar to that provided by a periodic chart of the elements.

REPLOT

Much of the screen display can come from a storage device containing plotted information from the computer. Over several minutes time, the image plotted on this device decays and produces a loss of quality in the student's screen display. Pushing the REPLOT key recreates the image on the storage device. Thus, the student can refresh his T.V. screen.

ARROW

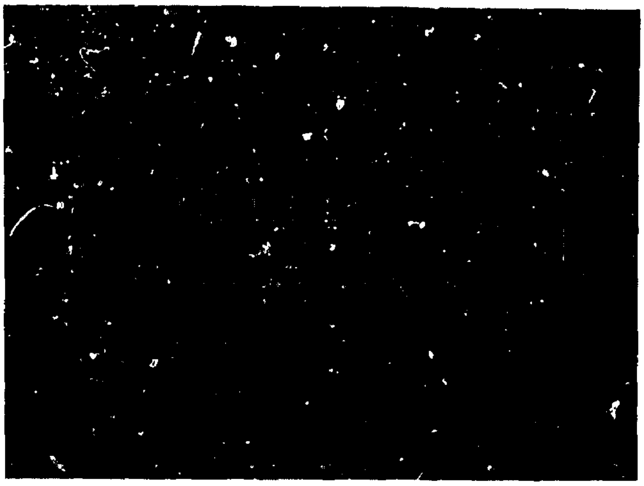
Several questions may occur in a given screen display. A small arrow appears on the screen near the particular question to which the student is currently responding. His typewritten response appears on the screen after this arrow. Initially, the student's response is directed to the first question in the display. However, he may choose to address another question on the display. By pressing the ARROW key, the student can move the arrow to whatever question he wishes to address.

The *set of arrows* on the left hand side of the keyboard can be used for a variety of author determined functions. A Geometry course developed at University High School in Urbana uses these keys to move a small dot around a grid shown on the screen. The dot moves in the direction of the arrow key pushed. Additional functional keys allow the student to mark a current dot by a large circle and to draw lines connecting marked dots. Thus, geometrical figures can be produced.

New authors need not fear that there are too many functional keys. They can use only the keys needed. Indeed, several lessons use only the NEXT key.

ILLUSTRATED LESSON SEGMENTS

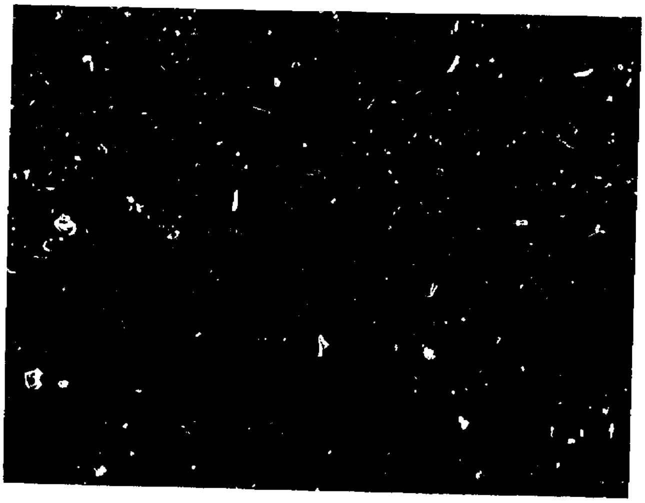
The remainder of this chapter illustrates with actual photographs parts of lessons taught on the PLATO computer system utilizing the TUTOR language.



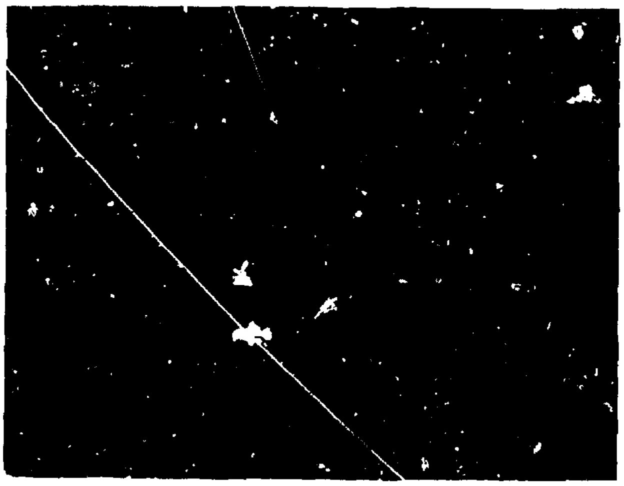
The above message appears on the T.V. screen of any student station not being currently used. A student begins conversation with the PLATO computer by pressing the NEXT key. He is then asked to write in his name. If PLATO recognizes him, his lesson resumes where he last left off. As an example, say that a student is studying French phonetics. The last time the student worked with PLATO he was in the middle of a timed translation drill.

He resumes study and types in an answer.

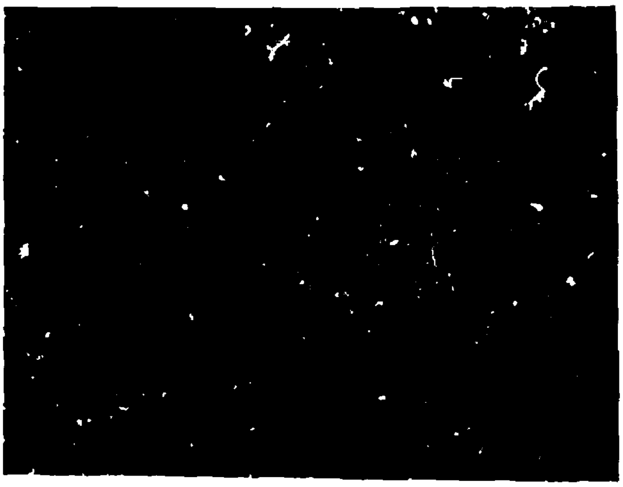
His wrong answer elicits help from PLATO.



Another student in a different course is asked a question about the discovery of the New World. His answer can be a phrase or sentence. PLATO evaluates this long answer by means of a keyword judger which is described in the next chapter.



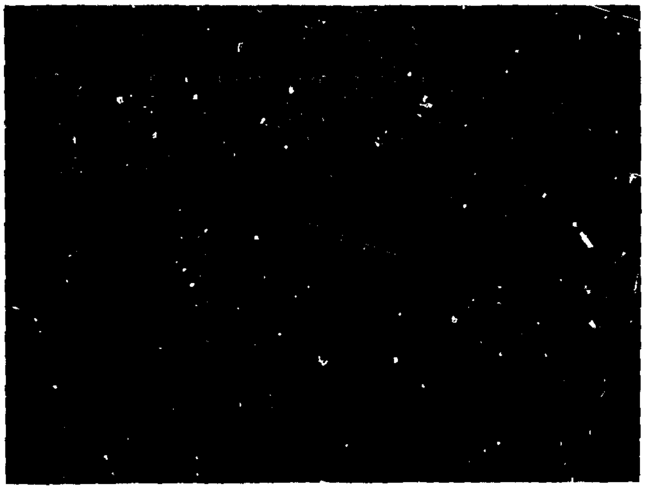
The student types in a long answer.



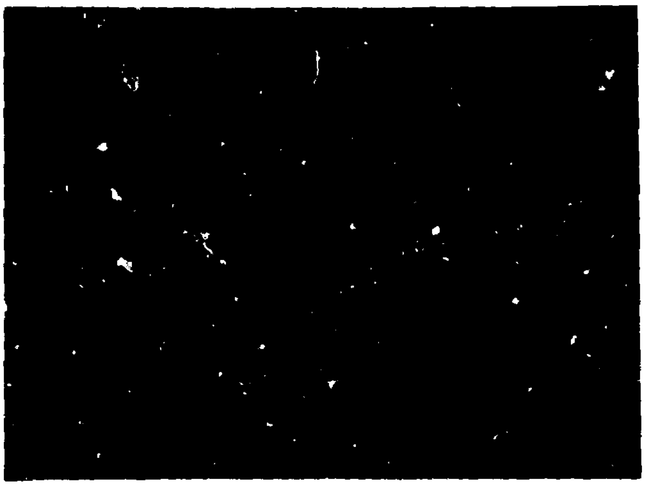
PLATO in turn underlines misspellings and crosses out unrecognizable words.

A sixteen lesson course in geometry used at University High School, Urbana, Illinois has been developed by J. Richard Dennis.* Dr. Dennis devised a grid system which allows the student to construct and evaluate geometrical figures. By storing relevant information during the student's construction of the figure, Dr. Dennis' lesson can distinguish the major geometrical figures regardless of their size, positioning or rotation on the grid. The student constructs a figure by using the set of arrow keys on the left hand side of the keyboard. These keys move a small cross in the direction of the arrow key pushed. The student can mark a location and upon marking a second location defines a line.

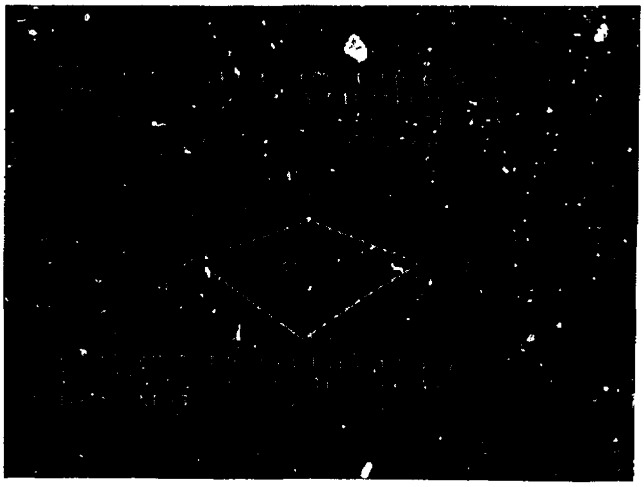
* Dennis, J. Richard. Teaching Selected Topics via a Computer System. CERL Report X-3; June, 1968. Urbana, Ill.



PLATO draws a line connecting the marks.



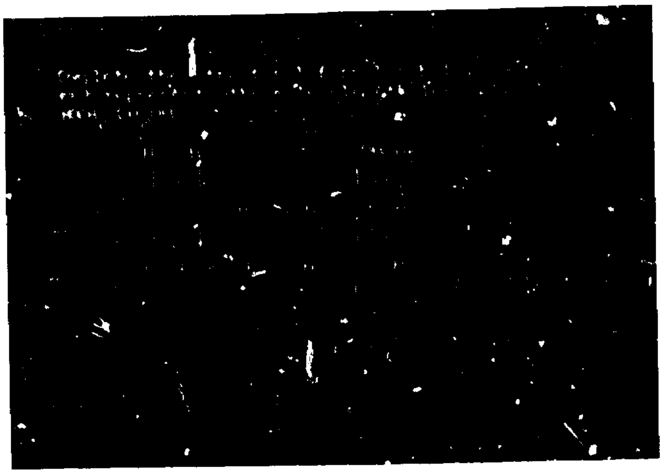
The student continues this process until his figure is complete.



PLATO then evaluates his correct figure and ...

asks him to draw the other quadrilateral possessing one line of symmetry. The student does so.

A last example demonstrates PLATO's usefullness in a beginning chemistry course at the University of Illinois.



Frequent mention has been made of the fact that the computer is directed by lesson authors in choosing what information will be displayed in a given situation and what evaluation will be given to a student respor,e. The next chapter describes how lesson authors can direct the PLATO computer by using the TUTOR language.

Chapter 3

HOW TO BEGIN LESSON WRITING

Lessons on the PLATO teaching system consist of a repeating sequence: a display on the student's T.V. screen followed by the student's response to this display. The display information may consist of slides, sentences, graphs--nearly anything of a pictorial nature--and in any combination. The student responds to this display by pressing a single key (e.g., the HELP or NEXT key) or by typing a word or sentence or even by making a geometrical construction. Lesson authors provide enough details about the possible student responses so that PLATO can maintain a dialogue with the student. The sequence of a display followed by a response is the building block of a TUTOR lesson and is called a UNIT.

An author constructs a lesson by writing one UNIT at a time. For each UNIT, the author specifies (1) the display that will appear on the student's T.V. screen, (2) how PLATO is to handle student responses to this display, and (3) how the current UNIT connects to other UNITs.

A statement written in the TUTOR language appears as follows:

```
WRITE     HOW  ARE  YOU  TODAY?
```

The first part of the statement (WRITE) is called the *command*, while the remainder (HOW ARE YOU TODAY?) is called the *tag*. Command names mnemonically represent PLATO functions. Following is a UNIT written in TUTOR. Figure 3.1 shows what a student would see on his T.V. screen while working on the UNIT.

```
UNIT     DAVINCI
WRITE    NAME THE ARTIST WHO
         PAINTED THIS PICTURE -
SLIDE    24
ARROW    1110
ANS      LEONARDO DA VINCI
WHERE    1301
WRITE    YOUR ANSWER TELLS ME THAT YOU
         ARE A TRUE RENAISSANCE MAN.
WRONG    WHISTLER
WHERE    1301
WRITE    I HOPE YOU ARE JOKING.
```

As one can infer, tags individualize commands for the particular function desired. The statements in this UNIT will be explained fully to verify inferences.

```
UNIT     DAVINCI
```

The UNIT statement initiates each UNIT. The tag (DAVINCI) will become useful later when UNITs are connected together to form a lesson. Each UNIT must have a name. No two UNITs may have the same name.

```
WRITE    NAME THE ARTIST WHO
         PAINTED THIS PICTURE -
```

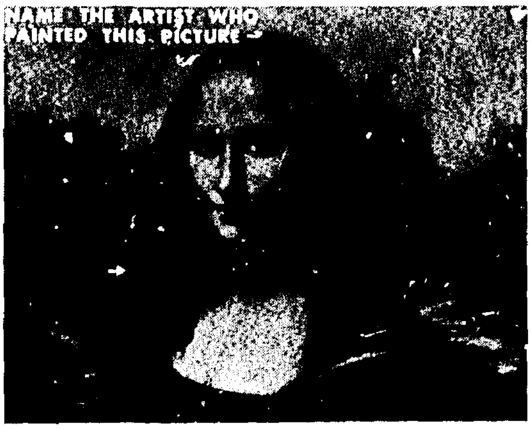
The WRITE statement causes the information contained in the tag to be displayed on the student's screen. The writing starts at the top left corner of the screen.

```
SLIDE    24
```

The SLIDE statement tells PLATO to show slide 24 on the student's screen. Slides and writing are superimposed on the screen. In this case, slide 24 is a picture of the beguiling smiler, Mona Lisa.

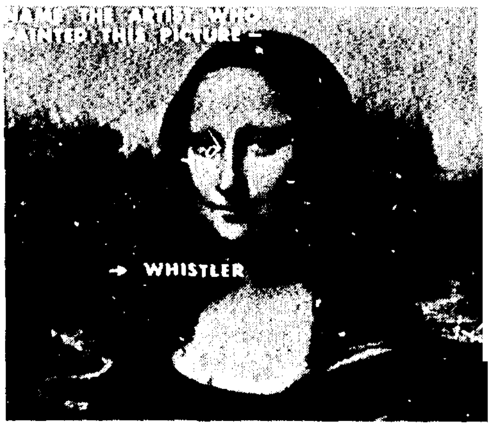
```
ARROW    1110
```

The ARROW statement acts as a boundary-line that separates preceding display statements from following response-handling statements. Thus, what precedes the ARROW command produces the T.V. display which remains on while the student works on the UNIT. Statements after the ARROW command are used in handling student responses to the display.


NAME THE ARTIST WHO
PAINTED THIS PICTURE –

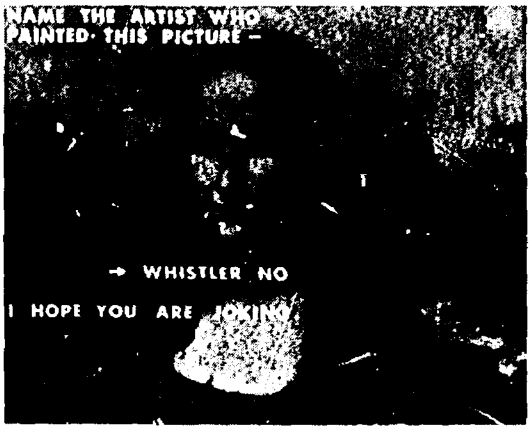
→

A. INITIAL UNIT DISPLAY


NAME THE ARTIST WHO
PAINTED THIS PICTURE –

→ WHISTLER

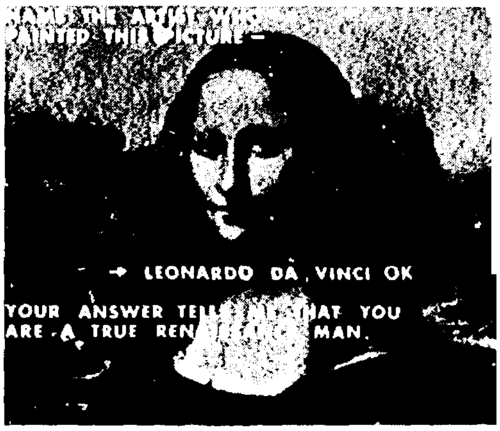
B. FIRST STUDENT RESPONSE


NAME THE ARTIST WHO
PAINTED THIS PICTURE –

→ WHISTLER NO

I HOPE YOU ARE JOKING.

C. PLATO'S JUDGMENT


NAME THE ARTIST WHO
PAINTED THIS PICTURE –

→ LEONARDO DA VINCI OK

YOUR ANSWER TELLS ME THAT YOU
ARE A TRUE RENAISSANCE MAN.

D. CORRECT RESPONSE AND JUDGMENT

Figure 3.1

The T.V. screen at four phases of a student's study of Unit DAVINCI

In addition, the ARROW statement notifies PLATO that a student response is required at this point in the lesson. Not only must an author leave room in the display for a student response, he must tell PLATO where that space is. The tag of the ARROW statement locates the student response on the screen. An arrow is shown on the screen at this place to tell the student where his response will appear. The tag 1110 is coded as follows. Consider the number 1110 as two pairs of numbers--11 and 10. The first pair refers to the line count and goes from 01 (the top line on the screen) to 18 (the bottom line on the screen). The second pair refers to the character count on the given line and goes from 01, the left side of the screen, to 48, the right side of the screen. Thus, 0101 refers to the first line first-character position, while 1848 refers to the last-character position on the bottom line. This convention for referring to screen positions is used in other TUTOR commands.

```
ANS      LEONARDO DA VINCI
 .
 .
 .
WRONG    WHISTLER
```

The ANS (mnemonic for answer) and WRONG statements are used to evaluate the student's response. If the response matches the tag of the ANS statement, PLATO writes "OK" after the student's response. "NO" is written for a match to a WRONG statement. An "OK" judgment allows the student to proceed to the next UNIT, whereas a "NO" judgment requires the student to erase and try again. Any response not foreseen by ANS or WRONG statements is judged "NO."

Having matched the student's response, PLATO proceeds to execute any display statements following the matched ANS or WRONG statement. Thus, student answers of "LEONARDO DA VINCI" and "WHISTLER" will receive appropriate responses from PLATO.

```
WHERE     1301
```

The WHERE statement indicates where the tag of the following WRITE statement will appear on the screen. The screen position convention already explained is used. Hence, PLATO's response to the student will start at the first-character position of line thirteen.

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

```
UNIT      DAVINCI
WRITE     NAME THE ARTIST WHO
          PAINTED THIS PICTURE -
SLIDE     24
ARROW     1110
ANS       LEONARDO
WHERE     1301
WRITE     THE COMPLETE NAME IS LEONARDO DA VINCI.
SPELL
ANS       LEONARDO DA VINCI
WHERE     1301
WRITE     YOUR ANSWER TELLS ME THAT YOU
          ARE A TRUE RENAISSANCE MAN.
WRONG     WHISTLER
WHERE     1301
WRITE     I HOPE YOU ARE JOKING.
WRONG
WHERE     1301
WRITE     HINT - MONA LISA - HINT
```

As you can see, any number of ANS and WRONG statements can be added to the response-handling section of the UNIT. Time and effort spent by an author in providing for student responses other than the common answer can greatly increase the ability to carry on a personal dialogue with each student. Use of the last WRONG statement (which has a blank tag) needs explanation. As previously mentioned, any unmatched student response is judged "NO." However, an author may wish to do something in addition to writing "NO" after an unanticipated response. The WRONG command with a blank tag facilitates such action and is called a "universal WRONG" statement. A student response that fails to match an ANS or WRONG statement tag is automatically "matched" to the universal WRONG statement. Display statements following this universal WRONG are then executed.

Thus, PLATO can give a hopefully appropriate comment even though the actual student response is not recognized (just as human teachers often try to do).

The SPELL command is also introduced here. In matching responses to ANS tags, PLATO uses a precision which often seems undesirable. Renderings of LEONARDO DA VINCI as LEANARDO DA VINCI or LEONARDO DAVINCI cause mismatches. Simply to judge these student responses "NO" would cause confusion. Is the concept incorrect or only the spelling? The SPELL command resolves this problem by telling PLATO to place "SP" after a student response if a slight rearrangement of the response would result in a match with following ANS tags. The student must correct the misspelling to continue.

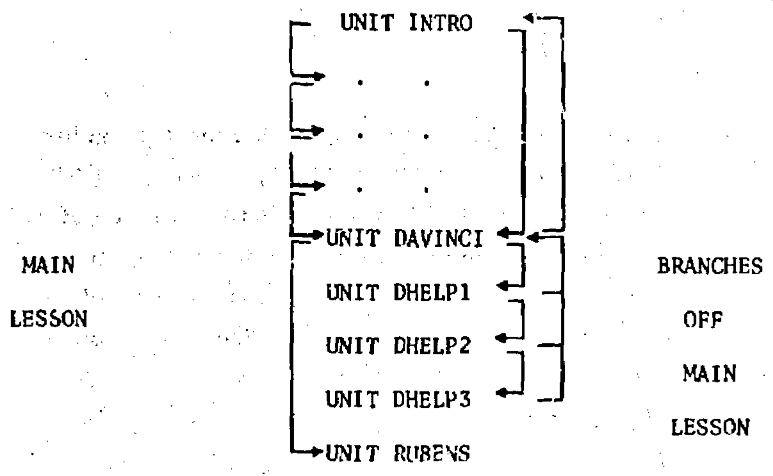
Lessons could be written using only the commands already discussed. Expository UNITs could be written using only display commands. Tutorial UNITs could be interspersed to test a student's understanding of the lesson material. Thus a simple linear chain of UNITs could form a lesson. However, mastery of a few more TUTOR commands opens up a wealth of "branching" possibilities. Branching, the technique of allowing alternate paths through a lesson, is the key to personal dialogue with each student. The example UNIT will therefore be expanded to include NEXT, BACK, and HELP commands.

```
UNIT     DAVINCI
NEXT     RUBENS
BACK     INTRO
HELP     DHELP1
WRITE    NAME THE ARTIST WHO
         PAINTED THIS PICTURE -
SLIDE    24
ARROW    1110
ANS      LEONARDO
WHERE    1301
WRITE    THE COMPLETE NAME IS LEONARDO DA VINCI.
SPELL
ANS      LEONARDO DA VINCI
WHERE    1301
WRITE    YOUR ANSWER TELLS ME THAT YOU
         ARE A TRUE RENAISSANCE MAN.
WRONG    WHISTLER
WHERE    1301
WRITE    I HOPE YOU ARE JOKING.
WRONG
WHERE    1301
WRITE    HINT - MONA LISA - HINT
WRONG    MICHELANGELO
NEXT     MREVIEW
```

The tag of the NEXT statement following the UNIT command gives the name of the next UNIT the student will see upon the successful completion of UNIT DAVINCI. The NEXT statement is necessary because in a highly branching lesson sequence the next UNIT for a student may not be the UNIT following in the write-up. For example, a diagram of the lesson flow involving UNIT DAVINCI might be:

PARTIAL DIAGRAM OF

LESSON ARTSY



The tag of the BACK statement gives the name of the UNIT the student will see upon pressing the BACK key. As you may infer, an author may choose to allow students to "backup" through the main lesson flow. However, the current example "backs up" to UNIT INTRO which might, for example, contain a list of the artists to be studied in the lesson.

The HELP statement refers to a help UNIT which the student may reach through use of the HELP key. Help UNITs are constructed in the same manner as UNIT DAVINCI. However, the last (or only)

UNIT in a help sequence is terminated by an END command. Upon completing the last HELP UNIT, the student is returned to the main UNIT from which he branched--in this case UNIT DAVINCI. Help UNITs for UNIT DAVINCI could appear as follows:

```
    UNIT     DHELP1
    SLIDE    25
    WRITE    HERE ARE SOME ADDITIONAL
             WORKS BY THE PAINTER OF
             THE SMILING LADY.
    UNIT     DHELP2
    SLIDE    26
    UNIT     DHELP3
    SLIDE    27
    END
```

As another example of TUTOR branching, consider the following situation. A student, working on UNIT DAVINCI, responds "MICHELANGELO." Previously, the student had worked his way through a series of UNITs concerning Michelangelo. The author therefore feels that the student must have missed something in the previous study and must be given further information about Michelangelo. The set of TUTOR statements

```
    WRONG    MICHELANGELO

    NEXT     MREVIEW
```

permit an author to force the student into additional material concerning Michelangelo. When the student responds "MICHELANGELO," he will see this answer judged "NO." He will not be able to erase. Instead, he can only go to UNIT MREVIEW. Upon completion of the Michelangelo review, which may consist of any number of UNITs, the author may return the student to UNIT DAVINCI. Thus, this student's lesson flow would consist of:

1. a series of UNITs on Michelangelo,
2. a question about the Mona Lisa; error leads to
3. a further study of Michelangelo, and
4. a return to the Mona Lisa.

Consider now the problem of using UNIT DAVINCI for a second student response. Additional display information is needed to ask the student a second question and another ARROW command is needed plus a second set of response handling statements. The UNIT may appear as follows:

```
UNIT     DAVINCI
NEXT     RUBENS
BACK     INTRO
WRITE    NAME THE ARTIST WHO
         PAINTED THIS PICTURE -
WHERE    1501
WRITE    IN WHAT CENTURIES DID THIS ARTIST WORK?
SLIDE    24
ARROW    1110
HELP     DHELP1
  .      }
  .      } Response-handling statements
  .      } for first arrow.
  .      }
ARROW    1601
HELP     DTIME
  .      }
  .      } Response-handling statements
  .      } for second arrow.
  .      }
```

One may wonder why the first WRITE statement in the UNIT is not preceded by a WHERE statement. The first WRITE statement automatically starts in the first character position on the top line of the screen. However, that assumption can be overridden by using a WHERE statement of your choice.

Notice that specific HELP statements are placed after each ARROW command. Placing the HELP statements in this location provides the student with help sequences specific to the question he is working on.

The second question, "In what centuries did this artist work?," gives rise to a large number of possible student responses which must be judged "OK." Students may respond "15 and 16," "the 15th and 16th centuries," "fifteenth and sixteenth," etc. Students may even respond with variations of "Leonardo Da Vinci worked from the fifteenth century to the sixteenth century." Hundreds of correct responses exist. To list all possibilities by means of ANS statements is clearly impractical and programming a computer to understand sentence syntax is currently unsolved. However, an attack can be made on this problem if one considers a sentence to consist of key words together with filler ones. Thus, the words "fifteen" and "sixteen" are the only essential words of the answer. "Century," "of," "the," and "and" are filler words. The following lesson segment illustrates how this division of words may be used to handle responses for the second question of UNIT DAVINCI.

```
    .
    .
ARROW    1601
HELP     DTIME
ANS      THE FIFTEENTH AND SIXTEENTH CENTURIES.
SPELL
MUST     15, 15TH, FIFTEENTH
MUST     16, 16TH, SIXTEENTH
DIDDL    THE, AND, CENTURY, CENTURIES, FROM, TO, HE, WORKED,
         LEONARDO, DA, VINCI
CANT     13, 13TH, THIRTEENTH, 14, 14TH, FOURTEENTH
WHERE    1801
WRITE    YOUR DATES ARE TOO EARLY.
CANT     17, 17TH, SEVENTEENTH, 18, 18TH, EIGHTEENTH
WHERE    1801
WRITE    YOUR DATES ARE TOO LATE.
```

A MUST statement contains an important word along with any acceptable synonyms for this word. A student response must include one of these words to be judged "OK." The author includes as many MUST statements as there are important words in the desired response. Failure of a student response to include one word from each "MUST" statement results in the student's response being judged incomplete. The DIDDL statement contains a list of words which may or may not occur in the student response. These words are ignored during judging. Any words in the response not accounted for in MUST and DIDDL tags are considered inappropriate and result in a "NO" judgment. CANT statements indicate a list of inappropriate words for which the author desires to take some specific action. The action is specified by statements following the CANT commands.

A collection of MUST, DIDDL, and CANT statements allows handling of student phrase or sentence responses. The price paid for this flexibility, however, is that the order of words in a student's response is not considered in the judgment. A student would be judged "OK" for his answer "LEONARDO DA 15TH WORKED SIXTEENTH VINCI CENTURIES."! The ANS statement is present to tell PLATO what to put on the screen of a student who pushes the ANSWER key.

Fifteen commands have been illustrated in this chapter. While over seventy TUTOR commands exist, most of the additional commands are as easy to master as those already explained. Mastery of the complete repertory of TUTOR commands is neither necessary nor sufficient to guarantee useful student lessons. Novice authors with clear goals can write useful lessons using only the commands already discussed. On the other hand, the most sophisticated programmer may write worthless lessons using the full set of TUTOR commands. The number of TUTOR commands mastered by an author should be dictated by the requirements of the lesson material and not by a desire to use all TUTOR commands in a lesson.

Chapter 4

HOW TO CUSTOM TAILOR YOUR LESSONS

A goal of good teaching is to tailor the instruction to the needs and background of individual students. If a student demonstrates failure to learn something, the teacher may try alternate approaches to the material. Similarly, when a student shows mastery of a topic, the teacher moves on to new material. Such flexibility is relatively easy in a tutorial situation (one teacher to one student) but more difficult in classroom instruction. Fortunately, PLATO allows a form of instruction which is quite close to the ideal tutorial situation.

In the last chapter you saw how the Unit, the basic element of a TUTOR lesson, was constructed and how Units may be connected together to form simple lesson segments. This chapter elaborates on the important subject of Unit interconnection. Before describing the TUTOR commands used to connect Units together, a common type of lesson framework will be examined.

Certain Units may be considered basic to the presentation of a lesson to a particular student. Such Units are called base Units. Each base Unit can be considered to be a decision point in a lesson. The student is either ready for the next major step in the lesson (a new base Unit) or he is not. If the student is not prepared for the next step, he is given supplementary material until he is prepared. A student's main lesson is defined as the path through his base Units. The first Unit in a lesson is automatically the first base Unit of each student's lesson. Following this first base Unit, each student moves to additional base Units. For each base Unit, a student may branch into Units supplementary to the base Unit. After a student goes into supplementary Units, he must return to the base Unit from which he started and resume the main lesson. PLATO's record of the current base Unit serves as a marker to facilitate the return from the supplementary Units. Unit interconnections within the supplementary Units do not reset the base marker. The marker is analogous to a bookmark which keeps the student's place in his text while he is using a reference book.

TUTOR branching commands can thus be divided into two categories:

1. Those which permit movement between base Units and,
2. Those which permit supplementary lesson sequences.

Commands Which Allow Movement Between Base Units

The NEXT statement specifies what base Unit the student will be sent to when he completes his current base Unit and presses the kev marked "NEXT". Usually the student is not allowed to use the NEXT key to move forward in the lesson until he has correctly answered all questions in a Unit. However, the author might want to select alternate Units contingent on a certain response by a student. For example, an incorrect answer which indicated a misunderstanding of a concept might best be followed by a few Units which give extra emphasis to the missed concept. As you may remember, there was an example like this in Chapter 3. The author can force the student to move on to these remedial Units before answering other items in the original Unit. A more complete explanation of such "contingent operations" as well as details of use of the NEXT command in these situations will be covered in Chapter 5.

Like the NEXT command, the JUMP command specifies a Unit which the student will be sent to. In our example of a response contingent operation of the NEXT command, the student entered a wrong answer and found that it was judged "NO" by PLATO. Having given this particular wrong answer, the student is permitted only to move on to a remedial sequence of Units. However, until he presses key NEXT he remains at the original Unit. The remedial Units would be seen only after the student pressed NEXT. In contrast, if a JUMP command had been used instead of the NEXT command the student would be sent to the first remedial Unit as soon as PLATO judged his answer "NO". The student would not have even seen PLATO write "NO" after his incorrect answer before the shift in Units took place.

Our discussion of the use of NEXT and JUMP branches as contingencies of particular student answers suggests a way in which these connection commands could be used to allow the student to select his own connections between Units. Suppose the student is working on Unit SELECT.

```
UNIT            SELECT
WRITE           SELECT A LESSON BY

                PRESSING THE APPROPRIATE KEY

                     PRESS KEY...     TO SEE...

                            1         ADDITION DRILL

                            2         SUBTRACTION DRILL

ARROW           1001
LONG            1
ANS             1
JUMP            ADDA
ANS             2
JUMP            SUBA
WRONG
WHERE           1101
WRITE           YOU MUST SELECT EITHER 1 OR 2,

                PRESS -NEXT- AND TRY AGAIN
```

You can see that Unit SELECT is very similar to the Units shown in Chapter 3. The statement "LONG 1" tells PLATO to "judge the students's answer as soon as it is 1 character long" (i.e. immediately after the student has pressed one key). If the student presses key 1, PLATO will immediately judge this "answer" and find that it matches the tag of the ANS 1 statement. Since there is a JUMP to Unit ADDA which is contingent on the student giving this particular answer, the student will be sent to Unit ADDA as soon as he presses key 1. A similar effect would have occurred if the student had pressed key 2 (except that he would have been JUMPed to Unit SUBA). If the student had pressed any other key his answer would have been judged "NO" by PLATO and he would have received the message "YOU MUST SELECT EITHER 1 OR 2, PRESS -NEXT- AND TRY AGAIN." Unit SELECT thus gives the student the voluntary choice of going to either the first Unit of an addition drill sequence or the first Unit of a subtraction drill sequence.

Commands for Branching to Supplementary Material

As you saw in Chapter 3, the HELP command specifies a Unit to which the student will be shifted whenever he presses the HELP key. The term "HELP" arises from a typical use of this type of branch. The DATA and LAB commands operate in a similar manner and provide additional branching possibilities for a student situation. The Units reached by the HELP, DATA, or LAB key contain supplementary information which is intended to aid understanding of the material in the main Unit. They are not limited to such use however. Mathematical tables, vocabulary lists, review sequences and a host of other reference-type material can be stored in HELP, DATA, or LAB Units for ready student access.

A HELP-type sequence may consist of as many Units as desired. The student moves through such a sequence just as if it were part of the main lesson. If the student presses the NEXT key after completing the last Unit of the sequence he will be returned to his base Unit in the main lesson. He may also return to his base Unit at any intermediate point in the sequence by pressing the SHIFT and BACK keys simultaneously. Each Unit in a lesson (including HELP-type Units) may have its own HELP-type sequences.

In other situations it might be necessary for students to have direct access to a great many small pieces of information. For example, suppose a lesson uses many new terms which the student may not be familiar with. It would be convenient if the author could give him the definition of any term upon request. This is essentially the effect of the TERM command. The TERM command specifies a term which will allow access to a single Unit. The student desiring information presses the TERM key. A message appears at the bottom of his TV screen asking "WHAT TERM?" When the student types a term, say "CAT," and presses the NEXT key he is sent to the Unit which contains a TERM statement with that particular term (e.g. CAT).

```
UNIT          DICT74
TERM          CAT
TERM          FELINE
WHERE         401
WRITE              CAT (FELIS CATUS)

              CARNIVOROUS QUADRUPED MAMMAL.

              FREQUENTLY KEPT AS A PET

              BY DISCERNING HUMANS.
```

The same Unit could be reached by typing any number of different terms so long as each term (e.g. FELINE) appeared as the tag of a TERM command in that Term Unit. Thus information can be "cross-referenced." When the student is finished with the Term Unit he presses the NEXT key and is returned to his original Unit.

Like the HELP branch, the TERM branch is not limited to the use suggested by the word "term." Any information which can be indexed by single words or short groups of symbols could be stored in Term Units. One convenient use of a Term Unit is as an index to other material. Unit TABLE allows access to the listed information from any other Unit in the lesson.

```
UNIT          TABLE

TERM          INDEX

WRITE         PRESS KEY...          TO SEE...

                      A                 HEART RATE

                      B                 TEMPERATURE

                      C                 RESPIRATION
ARROW         1001
LONG          1
ANS           A
JUMP          HRT
ANS           B
JUMP          TMP
ANS           C
JUMP          RESP
```

Notice that the JUMP command appears here despite its normal use for movement between base Units. Both JUMP and NEXT may be used within HELP or

TERM Units. However PLATO will not shift the "marker" which indicates the original base Unit. Thus, in the above example, Units HRT, TMP and RESP will be considered as supplementary Units rather than new base Units.

The BACK command specifies what Unit the student will be sent to if he presses the key marked "BACK" on his keyboard. The specified Unit may be the Unit that was last seen. Thus, the BACK key would act as a "reverse" allowing students to review previously covered Units. The student returns to his base Unit by pressing his NEXT key. Since the author has complete control over what Unit is specified, it is also possible to send students to special review Units. The author can even prevent use of the BACK key in particular Units simply by not including a BACK command.

In some situations, e.g. when using a Term Unit as an index to other material, the author may wish to redefine a supplementary Unit as a base Unit. Inclusion of a BASE command in the supplementary Unit will perform such a redefinition. Thus, Units reached through HELP, DATA, LAB, BACK, and TERM commands can become main lesson base Units.

Using Stored Student Information for Branching

So far, all of the described connection (or branching) operations are fixed at the time the lesson is written. The author may, however, want to allow alternate connections as a result of prior student performance. For example, the author may want to give additional explanations to students who make too many mistakes during the lesson. Connection commands such as NEXT, BACK, JUMP, TERM, HELP, LAB, and DATA, can use stored student information to "decide" what connections each student may make from each Unit. In short, the author is able to provide alternate branches from a Unit and let PLATO decide (from specified student information) which of the alternate connections each student will be allowed. This general technique of assigning branches based on stored student information will be fully explained in Chapter 7.

To review: TUTOR "branching" commands allow authors to prepare lessons which are responsive to the needs of individual students. Two of these commands (NEXT and JUMP) are used for connections between base Units. These two commands may also be used for connections between supplementary Units. Other commands (HELP, LAB, DATA, TERM, AND BACK) are restricted to connections to supplementary Units. However, the BASE command may be used to redefine a supplementary Unit as a base Unit. The author may provide alternate connections to various Units on the basis of specified student responses or stored student information. The quality and quantity of lesson individualization is limited only by the ingenuity of the author. The next few chapters will give you full details on how to incorporate these connection features into your own lessons.

Chapter 5

TUTOR COMMANDS

An Overview

All of the TUTOR commands can be arbitrarily placed in one of six categories based on their major function in a lesson. The only purpose in making such a classification is to insure that the new TUTOR author will not overlook a useful command simply because its function is not suggested by its title. The following list of categories (and the commands outlined within the categories) should be used mainly to suggest which commands warrant further investigation. The brief description of a command's function is oversimplified and based on typical use of the command. A more complete description of each command appears later in this chapter. The individual descriptions of the commands and the examples shown there will suggest more applications (as will practice in using the commands in actual lessons).

Do not be overwhelmed by the number of available commands. A great deal of useful lesson writing can be done using only about a dozen of the possible commands. Nor is it necessary for you to be familiar with every option of the commands you do use. Many basic commands (such as WHERE) offer a great deal of flexibility for the author who has special requirements but can be used in a much more restricted fashion by authors who do not need this flexibility. At the same time, your efforts to understand and use the complete range of TUTOR commands will be well rewarded.

(1) "Lesson" Commands

This group of commands is used in presenting the lesson as a whole. Some of these commands are used only once (or not at all) in a given lesson.

AREA	- gives lesson title and author identification
UNIT	- gives unique name to each segment of the lesson
END	- specifies the end of a main or help-type sequence in the lesson
T6ØTHS	- specifies that times of student responses will be recorded in 60ths of a second (usually recorded in minutes and seconds)

BASE - specifies that a supplementary segment of the lesson is to be redefined as a part of the main lesson.

UPLOW - makes both upper and lower case characters as well as special language character sets available in the lesson

C - allows author to annotate lesson

(2) "Display" Commands

These commands allow information to be shown on the student screen

WRITE or WRUSS - allows printing (using a standard set of characters) on screen

CHAR - allows special characters to be designed

PLOT - displays special characters designed by CHAR

LINE - displays straight line anywhere on screen

SLIDE - displays photographic material

SHOW - displays information stored during the lesson

WHERE - specifies where on the screen WRITE, PLOT, or SHOW information is to be presented

INHIB - inhibits certain standard TUTOR displays for special requirements

(3) "Response" Commands

These commands specify where student responses are expected in a lesson and how they are evaluated.

ARROW - indicates that a student response is required

ANS or ANSRU - specifies a single response which will be accepted

WRONG or WRGRU - specifies a single response which will not be accepted

SPELL - checks the spelling of a student's answer

JUDGE - evaluates a response (overrides prior judging)

BUMP - allows PLATO to ignore specific single characters which are irrelevant (e.g., spaces or certain punctuation) in a particular response

RESET	- cancels prior judging options and resets judging to standard form
MUST	- specifies words which must appear in a correct sentence-type answer
CANT	- specifies words which must not appear in a correct sentence-type answer
DIDDL	- specifies words which can be ignored in a correct sentence-type answer
PUT	- allows interpretation of specified single characters as equivalent to other specified characters
NODUP	- allows duplicate student responses to be rejected
LONG	- allows responses of a specified length to be judged automatically
TIME	- limits the time a student has to give a responsc

(4) "Branching" Commands

These commands allow the author to specify the order in which units will be arranged.

NEXT	- specifies what Unit is next in the lesson (overrides linear order)
BACK	- specifies what Unit the student will go to if he presses the "BACK" key
GOTO	- gives Unit alternate forms depending on certain stored information
JUMP	- forced branch based on a specific student response or value of certain stored information
HELP (DATA or LAB)	- allows voluntary (and temporary) branch to specified Units by using special keys
TERM	- specifies information to be received on a voluntary (and temporary) branch using the "TERM" key

(5) "Splicing" Commands

This command allows information which is used in many parts of the same lesson to be written only once and "spliced" in wherever needed.

JOIN	- allows insertion of statements which are identical to those appearing in a specified Unit in the same lesson

(6) "Calculation" Commands

These commands allow numerical and logical operations by the student and the author.

CALC	- performs a mathematical operation as a part of the lesson
ICALC	- similar to CALC but performs more restricted operations at a much higher speed
FCALC	- similar to CALC but performs somewhat more restricted operations at a higher speed
STORA	- allows student to use his PLATO station as a desk calculator
STORE	- stores a student response (which may be alphabetic as well as numerical)
ACALC	- stores alphanumeric information
INFO	- makes a record of specified stored information on magnetic tape
CLOCK	- stores the amount of time elapsed since the student signed in for the lesson
ADD1	- increases the value of a stored number by one
SUB1	- decreases the value of a stored number bv one
ZERO	- sets a stored number to zero
RANDP	- stores a number selected at random (without replacement) from a list of integers
IPERM	- sets up a list of intergers for RANDP
RANDU	- stores a number selected at random from a uniform distribution
LOOP	- allows multiple operations in excess of the usual limit

Contingencies

Suppose that a student is given a problem. More specifically, let us suppose that the problem consists of a statement followed by several questions which must each be answered by the student. There are four periods during the presentation of such a task that we might want PLATO to do particular operations. These periods are:

1. when the student obtains the initial presentation,
2. when the student selects a question to answer,
3. when the student answers the question and requests PLATO's evaluation, and
4. after PLATO has evaluated the answer.

For example we might want to:

1. display the problem and record the time at which the student first saw the problem,
2. show a special message related to each question,
3. ignore certain irrelevant parts of a student's answer, and
4. send the student to a special review unit for certain wrong answers.

Each of these last four operations are <u>contingent</u> on PLATO's being involved in one of four basic functions of lesson presentation. They are referred to as UNIT contingencies, ARROW Contingencies, JUDGE Contingencies and ANSWER-TYPE (or ANS) Contingencies.

Unit, Arrow, and Answer Contingencies

In terms of some of the basic TUTOR commands which you have already seen, we might want to present a message to the student by means of a WRITE command

1. when he enters a Unit
2. when he selects a particular question to answer
3. when his answer is recognized.

This is done in the TUTOR UNIT below.

```
UNIT        QUES8
WRITE       ANSWER THESE PROBLEMS

            3 + 3 =

            3 X 3 =
ARROW       309
WHERE       701
WRITE       TAKE YOUR TIME
ANS         6
WHERE       320
WRITE       VERY GOOD
ARROW       509
WHERE       701
WRITE       KEEP CALM
ANS         9
WRONG       6
WHERE       520
WRITE       MULTIPLY, DO NOT ADD
```

When the student enters UNIT QUES8 he sees the message

```
            ANSWER THESE PROBLEMS

            3 + 3 = →

            3 X 3 =

            TAKE YOUR TIME
```

which was produced by the WRITE command following UNIT QUES8 and the WRITE command following ARROW 309. Upon entry to the Unit the student has the question specified by the first ARROW command selected for him automatically. If he decided to try the other question first he could press the ARROW key and he would see the message

```
            ANSWER THESE PROBLEMS

            3 + 3 =

            3 X 3 = →

            KEEP CALM
```

produced by the WRITE command following UNIT QUES8 and the WRITE command following ARROW 509. In each of these cases the small arrow indicates where the student's answer will appear. The position of the small arrow is specified by the number following the ARROW command. Thus ARROW 309 puts a small arrow on line 3 in the 9th space and ARROW 509 puts a small arrow on line 5 in the 9th place. The ARROW command indicates that a response is required from the student and states where the response will appear on the screen.

In this example the message produced by the WRITE command following the UNIT command is known as a UNIT Contingency, or UNIT-C for short. Notice that this writing occurred when either of the two questions were selected. In general, commands placed after a UNIT command and before the first ARROW command in a Unit (or the next UNIT command if there are no ARROWs in the UNIT) are activated as soon as the student enters a Unit and stay activated (unless specifically overriden by later commands) as long as the student remains in that Unit. All such commands are termed UNIT Contingencies (or UNIT-Cs).

The messages produced by the WRITE commands following each of the ARROW commands are known as ARROW Contingencies, or ARROW-Cs for short. In general, commands placed after an ARROW command and before the first answer-type (ANS-type) command (any command which specifies a correct or incorrect answer is considered an ANS-type command) will be activated only while that particular ARROW is selected. All such commands are termed ARROW Contingencies (or ARROW-Cs).

Now suppose our student answers the first question by typing the number 6. After he requests that PLATO judge his answer (by pressing key "NEXT") his screen will show

```
ANSWER THESE PROBLEMS

3 + 3 = → 6 OK     VERY GOOD

3 X 3 =

TAKE YOUR TIME
```

As you might have guessed, the message "VERY GOOD" which was produced by the WRITE command following ANS 6 is an ANSWER Contingency, or ANS-C for short. The "OK" is produced automatically when a student's answer is judged and found to match an answer listed as being correct. There are several ANS-type commands. In general commands placed after any ANS-type command and before the next ANS-type command, ARROW, or UNIT command are activated when the tag of that particular ANS-type command matches the student response. The student response may be incorrect, i.e. the matched ANS-type command might be the command WRONG. For example, after answering the first ARROW correctly the student might answer the second ARROW with the number 6. He would then see

```
ANSWER THESE PROBLEMS

3 + 3 =   6 OK

3 X 3 = → 6 NO     MULTIPLY, DO NOT ADD

KEEP CALM
```

Notice that the prior ARROW-C and ANS-C writing is replaced by the current ARROW-C and ANS-C message. If the student erased his incorrect answer the "NO" and the "MULTIPLY, DO NOT ADD" messages would also disappear. This is logical since when the student erases an answer, the ANS-C comments no longer apply.

To review, a given command is part of a...

(1) UNIT-C	if it occurs after a UNIT command and before any ARROW commands in the Unit,
(2) ARROW-C	if it occurs after an ARROW command and before the first following ANS-type command,
(3) ANS-C	if it occurs after an ANS-type command and before the next ANS-type command (if any) or before the next UNIT command.

The operation specified by a command which occurs in a...

(1) UNIT-C	is activated when the student first enters the Unit and, if a display command, remains active during the entire time the student remains in that unit,

(2) ARROW-C	is activated whenever the student selects the given ARROW and, if a display command, remains active while the student is on that ARROW,
(3) ANS-C	is activated whenever the student's response matches the tag of an ANS-type command.

All contingent operations are terminated as soon as the student enters a new Unit.

Judge Contingencies

Not all TUTOR commands fit into this pattern of UNIT-Cs, ARROW-Cs or ANS-Cs. The most obvious of these are the UNIT and ARROW commands which are used to specify part of the boundaries of such contingencies. These special commands actually define contingencies and serve to form the basic structure of a lesson. In practice you must examine the individual description of each command to determine if it can be used under a particular contingency and, if so, what its effect will be.

The ANS-type commands, which are used to define the remaining boundaries for the ARROW-Cs and ANS-Cs, fall into our fourth and final category of contingencies. This remaining contingency occurs during the period between the time judging of a response begins and the time judging is completed. Usually judging begins after a student enters a response and requests that it be judged (by pressing key "NEXT"). Judging usually ends when a match to the student's response is made to the tag of an ANS-type command or no match can be made (and an automatic "NO" judgment is given). There are situations where we might want to alter this usual process of judging. This is done by a special group of commands which operate solely under the judging contingency (JUDGE-C for short).

For example, we might want to indicate to the student that he almost matched a correct answer. The SPELL command will alter the standard judging so that if the student's answer differs from an accepted answer by only a few letters, the letters "SP" (for "spelling") will be placed after the student's response instead of the usual "OK" or "NO" judgement.

As another example, we might not care about the manner in which the student separates a sequence of numbers which make up the proper response. We want to allow the student to use spaces, dashes, commas or any other unambiguous means of separating the numbers from one another. One way to allow this is to list as answers the correct string of numbers separated by spaces, the same numbers separated by commas, separated by the word "and," and so forth. A far better approach would be to ignore any spaces, commas, dashes, etc. when judging the student's answer. This can be done by the command BUMP whose tag contains the list of characters to be ignored.

Commands which act as JUDGE-Cs are located <u>after</u> the ARROW for which the judging is to apply and <u>before</u> the next ARROW (if any) in the Unit or the next UNIT command. At first thought you might expect there to be confusion between the function of commands in these locations since ARROW-C and ANS-C commands are also located there. However, any command which can function as a JUDGE-C cannot function under any other contingency and any command which can function as ARROW-C or ANS-C cannot function as a JUDGE-C.

Commands such as SPELL, BUMP, ANS, WRONG, etc. tell PLATO how to go about <u>judging</u> a response. Other commands, such as WRITE, have nothing to do with the operation of evaluating the student's answer. From another viewpoint, commands such as SPELL, BUMP, ANS, WRONG, etc. perform no function <u>unless</u> judging is in progress. They could thus not serve as an ARROW-C, which is executed before a response is given, or an ANS-C, which is executed after a response has been judged to be of a given type. Ultimately, the best (and easiest) way to determine the proper function of a command is to read the description of that command located at the end of this chapter. These descriptions specify under which contingency or contingencies each command may be used.

Once judging of a response to a particular ARROW is begun, each of the commands following the ARROW command is examined. Commands other than JUDGE-C commands are ignored. When an ANS-type command is found the tag is matched against the students response. Judging halts as soon as an exact match is found. If the command is another type of JUDGE-C, judging is altered in compliance with the directions of the JUDGE-C command and judging is continued to the next command. You can see that it is possible (and often desirable) for the author to alter the judging several times if necessary before a match is found. For example, consider an ARROW for which the correct answer is the number 23. Say that the

```
ARROW      201
WHERE      220
WRITE      SECOND ITEM
ANS        23
SPELL
BUMP        -
ANS        TWENTYTHREE
ARROW      301
```

student has typed "TWENY THREE" and pressed key "NEXT" to request that this answer be judged. PLATO begins checking all the commands following ARROW 201. The WHERE and WRITE commands form an ARROW-C (which was activated when ARROW 201 was first selected), hence, they are ignored. ANS 23 is an ANS-type command so PLATO checks to see if "23" matches the student's response (TWENY THREE). Since it does not match, PLATO continues to the next command, SPELL, which is a JUDGE-C that tells PLATO to alter its judging to accept slight mismatches as possible misspellings. The following command, BUMP, is also a JUDGE-C and tells PLATO to discard spaces and dashes in the student's response. At the next ANS command PLATO can thus look at "TWENY THREE" and identify it as a possible misspelling of the acceptable answer "TWENTYTHREE." The student would then see the letters "SP" placed beside his answer. Because of the BUMP command, PLATO would have accepted either "TWENTYTHREE", "TWENTY-THREE" or "TWENTY THREE" as correct responses.

Up to this point the terms UNIT-C, ARROW-C, etc. have been used to mean a single command whose activation was contingent upon PLATO being involved in one of four basic functions of lesson presentation. From now on these same terms may also refer to a group of commands which are similarly contingent

TUTOR Variables

In the first section of this chapter several references were made to "stored information" or "stored numbers." For example, it might be necessary to keep a record of the first name of each student so PLATO could say "you are doing very well, George" at some appropriate point in a lesson. In order to know _if_ PLATO should tell George that he was "...doing very well..." it would also be necessary for PLATO to keep track of how many wrong answers George has given during the lesson. Such information as the name "George" and the number "12" (George's total wrong answers) can be stored during a TUTOR lesson in storage spaces known as "TUTOR variables."

Chapter 6 will give you a more detailed description of how TUTOR variables work and how to use them. For now it will be enough to know that each student has 63 such storage spaces (variables) which can be referred to in a TUTOR lesson. Three types of information can be stored in these variables:

1) groups of alphabetic symbols like "GEORGE", "TEST 42", "JULY 29", "WHAT?", etc.
2) integers ("whole numbers") like "12", "1984", etc.
3) numbers with decimal fractions like "12.0", ".002", "45.7324", etc.

The type of information stored in a TUTOR variable is indicated by a "format code letter" which precedes the identification number of the variable. For example,

A20 indicates that TUTOR variable 20 contains _A_lphabetic or "word" information,

I32 indicates that TUTOR variable 32 contains an _I_nteger number,

F63 indicates that TUTOR variable 63 contains a number with a decimal _F_raction. (If you have computer programming experience, you may prefer to remember this as a _F_loating point number.)

When a TUTOR variable is used (say to store information received from a student) the author defines the format of the variable to match the type of information being stored. Thus, if a student is asked to type his first name and we wish this name to be stored in TUTOR variable 20, we would use the statement "STORE A20" in our lesson. On the other hand, say we wanted to use variable 20 to make a note of his answer to the problem "2X5= ." The command "STORE I20" would be used to store his response, since the expected answer would be an integer. Finally, if we wanted to use variable 20 to store the student's answer to a problem which would involve a decimal fraction, we would use the statement "STORE F20" in our lesson. Although expresions like "variable I20" and "variable F20" are often used for convenience, remember that "I20" and "F20" actually refer to only one variable.

PLATO uses the TUTOR variables of each individual student whenever a reference to a TUTOR variable is encountered in a lesson. Thus one student might be sent to a remedial Unit because his TUTOR variable 30 shows that he has made an excessive number of mistakes in a review test. Another student on the same lesson might be allowed to continue because his variable 30 shows a small number of mistakes.

A TUTOR variable with an "A" format can contain up to 8 symbols--letters, numbers or punctuation marks. Whatever letters, numbers, etc. are contained in a variable having an "A" format, it is important to remember that they will be treated <u>only</u> as symbols. Thus it would be possible to store a number like "125" in TUTOR variable 15 with an "A" format but the author could not do the same meaningful arithmatic operations that he could do if "125" had been stored with an "I" format. You might say that PLATO does not recognize that the symbols "1", "2", "3", etc. represent numbers when they are stored with an "A" format in a TUTOR variable. We will have more to say about the different formats used by TUTOR variables in Chapter 6.

Individual Commands

The remainder of this chapter consists of single page descriptions of each of the TUTOR commands. Each description consists of six sections:

(1) COMMAND:	The command itself. The command specifies a given type of operation to be activated during a lesson.
(2) TAG:	The additional information (if any) used with the command to specify the exact operation to be performed when the command is activated during the lesson.
(3) OCCURRENCE:	The contingencies under which the command can be used, i.e. UNIT-C, ARROW-C, ANS-C, JUDGE-C or SPECIAL. Earlier in this chapter there is a complete discussion of the idea of TUTOR contingencies. In the case of SPECIAL commands, the OCCURRENCE section explicitly describes the situation under which the command is used.
(4) EFFECT:	a description of what occurs when the command is activated.
(5) COMMENTS:	special notes about the use or restrictions in the use of the command.
(6) EXAMPLE:	a demonstration of the use of the command in a TUTOR lesson.

The next page shows the TUTOR commands in alphabetical order listed with the contingencies under which they can operate.

Alphabetic Index of TUTOR Commands

Command	UNIT-C	ARROW-C	ANS-C	JUDGE-C	SPECIAL
ACALC	X	X	X		
ADD1	X	X	X		
ANS				X	
AREA					X
ARROW					X
BACK	X				
BASE	X				
BUMP				X	
C					X
CALC	X	X	X		
CANT				X	
CHAR					X
CLOCK	X	X	X		
DIDDL				X	
END					X
FCALC	X	X	X		
GOTO	X	X	X		
HELP.	X	X	X		
ICALC	X	X	X		
INFO	X	X	X		
INHIB	X	X			
IPERM	X	X	X		
JOIN					X
JUDGE			X		
JUMP	X	X	X		
LINE	X	X	X		
LONG				X	
LOOP					X
MUST				X	
NEXT	X		X		
NODUP			X		
PLOT	X	X	X		
PUT				X	
RANDP	X	X	X		
RANDU	X	X	X		
RESET				X	
SHOW	X	X	X		
SLIDE	X	X	X		
SPELL				X	
STORA				X	
STORE				X	
SUB1	X	X	X		
TERM					X
TIME	X	X	X		
T6ØTHS					X
UNIT					X
UPLOW					X
WHERE					X
WRITE	X	X	X		
WRONG				X	
ZERO	X	X	X		

COMMAND:	ACALC
TAG:	A TUTOR alphanumeric variable followed by an equal sign and up to 8 alphanumeric characters. Indirect referencing (Chapter 8) is permitted.
OCCURRENCE:	UNIT-C, ARROW-C, ANS-C
EFFECT:	The characters to the right of the equal sign are stored in the variable on the left of the equal sign.
COMMENTS:	This command allows the author to store any combination of up to 8 characters in a TUTOR variable. It is particularly useful in providing labels for student data produced by the INFO command.
EXAMPLE:	This Unit produces data records which summarize a student's performance on a previous test. The number of correct answers is stored in I12; the number of wrong answers is stored in I13. Note that preceding blank characters are used in the ACALC command so the labels will be lined up with the "I" format numbers beneath.

```
UNIT  SUMMARY
ICALC I14 = I12 + I13
ACALC A25 =   TOTAL
ACALC A26 =   RIGHT
ACALC A27 =   WRONG
INFO  A25, A26, A27
INFO  I14, I12, I13
```

The data records below indicate the type of output Unit SUMMARY would produce.

```
JONES  35*26  SUMMARY  1  INFO  TOTAL     RIGHT     WRONG

JONES  35*27  SUMMARY  1  INFO     75        64        11
```

COMMAND: ADD1

TAG: A single TUTOR integer variable. Indirect referencing (chapter 8) is permitted.

OCCURRENCE: UNIT-C, ARROW-C, ANS-C.

EFFECT: Increases by 1 the value of the variable listed in the tag.

EXAMPLE: In this example ADD1 is used as an answer contingency controlling I5 (which is used here as a "corrects" counter). Notice in UNIT LAST that the WHERE command gives the beginning position of an 8 character field. The values of I5 and I6 will appear at the right of this field.

```
    .
    .
UNIT    BEGIN
ZERO    I5
ZERO    I6
WRITE   THIS LESSON CONSISTS OF 35
        PROBLEMS

    .
UNIT    PROB1
WRITE   WHAT IS...
ARROW   1010
ANS     ..
ADD1    I5
JUMP    PROB2
WRONG
ADD1    I6
JUMP    PROB2

    .
    .
END
UNIT    LAST
WRITE   YOU ARE NOW FINISHED.
        OF THE 35 PROBLEMS
        YOUR SCORE IS...

        CORRECT...
        WRONG.....
WHERE   504
SHOW    I5
WHERE   604
SHOW    I6
    .
    .
    .
```

COMMAND: ANS

TAG: The author's answer (a "correct answer")

OCCURRENCE: JUDGE-C

EFFECT: While judging, if the computer matches a student's response with the author's ANS tag, an "OK" will be placed after the student's answer. Then any Answer-type contingencies following the ANS command are performed.

COMMENTS: Any number of ANS or WRONG commands can be placed after any Arrow. An ANS command with a blank tag is termed a "universal answer" and will cause any answer that does not match any of the other ANS or WRONG tags to be scored "OK". All arrows must have "OK" responses before the student is permitted to go to the next UNIT. The tag of the first ANS command after the Arrow provides the "correct" answer that the student sees if he presses the ANS key. If answers will exceed about 20 characters see the discussion for LONG. Where the student's answer is given in Cyrillic characters, the command ANSRU should be used instead of ANS.

EXAMPLE: On the first question either "FOUR" or "4" is accepted as "OK". The second question will accept anything but "PURPLE" as correct. If "FOUR" is given on the first question the WRITE tag appears on line 17 (as an Answer contingency).

```
WRITE         WHAT IS 2+2?
              WHAT COLOR ARE YOUR EYES?
ARROW         113
ANS           4
ANS           FOUR
WRITE         THAT'S CORRECT BUT WHY SPELL IT OUT?
ARROW         226
ANS
WRONG         PURPLE
WRITE         I DONT BELIEVE YOU
   .
   .
   .
```

COMMAND:	AREA
TAG:	Up to 32 characters.
OCCURRENCE:	First command in the first Unit of a lesson.
EFFECT:	Provides a lesson title which allows identification and selection of a lesson by authors and system personnel.
COMMENTS:	Authors may test any lesson which has been read into the computer by "signing in" under the name "STUDENT." After signing in this way (or with student records which list no Lesson name), the author sees a choice table listing the AREA tags of all Lessons which are then in the computer. The author can thereby enter a lesson.

EXAMPLE:

```
UNIT    GEOM1
AREA    TRIANGLE EVALUATION BY G. P. BURDELL
C       GEORGE P. BURDELL
        APRIL 1, 1969
        .
        .
        .
```

COMMAND: ARROW

TAG: 1. A 4-digit number specifying a point on the screen. The first 2 digits specify one of 18 lines (01 through 18) and the second 2 digits specify one of 48 spaces (01 through 48) on this line. Thus 0148 specifies a point in the upper right corner of the screen. The first of these 4 digits may be omitted if it is zero, i.e., 0101 and 101 both specify the upper left corner of the screen.

2. When finer control of location is desired, a tag consisting of two numbers (separated by a comma) can be used. The first number specifies one of 170 vertical positions (0 is the top position). The second number specifies one of 240 horizontal positions (0 is the left of the screen). Standard characters used in TUTOR are written within a rectangle 10 units high and 5 units wide. Thus, to position an arrow between lines 10 and 11 and between spaces 4 and 5 (in terms of "single-number" coordinates) the statement ARROW 95,17 would be used.

OCCURRENCE: After all UNIT contingencies

EFFECT: An arrow is displayed at the point specified by the tag. The first ARROW command in a UNIT marks the end of Unit contingencies in that UNIT. Each ARROW command initiates any Arrow and Judge contingencies that might be present.

COMMENTS: Up to 20 ARROW commands can be used in each Unit. Student responses are displayed to the right of the arrow on the screen. Some type of answer command must follow every ARROW command for use in the Judge contingency. The student may answer any "arrow" first by pressing the "ARROW" key until the desired arrow is selected. Generally the student must satisfactorily answer all "arrows" before proceeding. Only one arrow appears at a time.

EXAMPLE: In the example below the first WRITE is a UNIT contingency while the other WRITE commands are Arrow contingencies and occur only when the specified arrow is selected. Note that the two WHERE tags have the same effect. The student's response appears at 0204 for the first ARROW and 0213 for the second since a space is automatically inserted after the arrow.

```
UNIT     TEST4
WRITE    PUT AN X BESIDE 1 AND A Y BESIDE 2
         1       2
ARROW    0202
WHERE    0301
WRITE    PUSH THE X KEY
ANS      X
ARROW    211
WHERE    301
WRITE    PUSH THE Y KEY
ANS      Y
```

COMMAND: BACK

TAG: A UNIT name. Assigned Operations (Chapter 7) and Indirect Referencing (Chapter 8) are permitted.

OCCURRENCE: UNIT-C

COMMENTS: When the BACK key is pressed by the student, he is shifted to the UNIT specified by the BACK command tag. If the student presses BACK again, he is shifted to the UNIT mentioned in a BACK command in this "BACKUP" UNIT. This process continues until the student reaches a UNIT lacking a BACK command. When the student presses the SHIFT AND NEXT key in any of the BACKUP UNITS, he will immediately return to the MAIN UNIT he was working on.

EXAMPLE:

```
    UNIT      ST1
    AREA      STUDY ONE
    BACK      NOBACK
    WRITE     HELLO.  TODAY WE SHALL...
      .
      .
    UNIT      ST2
    BACK      ST1
    WRITE     ...
      .
      .
      .
      .
    END
    UNIT      NOBACK
    WRITE     SORRY, THERE IS NO
              BACKUP FOR THIS PAGE.
      .
      .
      .
```

COMMAND: BASE

TAG No tag is used

OCCURRENCE: UNIT-C

EFFECT: Defines Unit as the student's base Unit regardless of how the student got to that Unit

EXAMPLE: This Unit permits a student to select various sub-lessons from any point in a lesson or sub-lesson by pressing key TERM and typing "CHOOSE". The student's restart records will reflect his actual position in the lesson (i.e., Unit SELECT) rather than the Unit from which Unit SELECT was entered.

```
UNIT        SELECT
TERM        CHOOSE
BASE
WRITE       PRESS KEY...     FOR
            A                VECTOR ADDITION LESSON
            B                VECTOR SUBRRACTION LESSON
            C                VECTOR MULTIPLICATION LESSON
ARROW       1020
LONG        1
ANS         A
JUMP        ADDA
ANS         B
JUMP        SUBA
ANS         C
JUMP        MULTA
ANS
JUDGE       IGNORE
```

COMMAND: BUMP

TAG: A list of characters to be ignored in the answer.

OCCURRENCE: JUDGE-C

COMMENT: The characters designate in the tag are "bumped" from the student's answer for judging. Thus, if some characters are irrelevant but may appear in a student's answer, they may be ignored in his answer during judging. Although the characters are "bumped" for judging, the student's answer on the screen remains untouched. If a space is to be "bumped", it must be the first character in the tag. Note that every character in the list will be bumped. Therefore you should not include anything (such as commas, dashes, etc.) in the list which you don't want ignored.

EXAMPLE:

```
     .
     .
     .
UNIT     MATH8
WRITE    WHAT ODD NUMBERS ARE
         BETWEEN 1 AND 8?

ARROW    510
BUMP      ,AND
ANS      357
     .
     .
     .
```

COMMAND: C

TAG: A message with any number of 60 character lines.

OCCURRENCE: Anywhere in a lesson.

COMMENTS: Messages in the tag of a C command will appear only in the printed copy of a lesson or during on-line editing of the lesson. These messages will not affect student operation. The main use of the command is for annotation of lessons for systems and author use. Months after a lesson is written, these notes will remind the author and inform new programmers why something was done the way it was.

EXAMPLE: This Unit uses the C command to identify the lesson author and indicate the use of a variable.

```
UNIT    A1
AREA    BURDELL'S LESSON
C       GEORGE P. BURDELL
        APRIL 1, 1969
ZERO    I40
C       I40 CONTAINS TOTAL CORRECT ANSWERS
```

COMMAND:	CALC
TAG:	A TUTOR variable followed by an equal sign and an arithmetic expression. The arithmetic expression may consist of constants, TUTOR variables, and operation symbols. All of the constants and TUTOR variables may be either integer or floating point. Indirect referencing (Chapter 8) is permitted
OCCURRENCE:	UNIT-C, ARROW-C, ANS-C
EFFECT:	The value of the arithmetic expression is placed in the variable to the left of the equal sign. Operation symbols permitted are:

+	(addition)	/	(division)	C	(cosine)
-	(subtraction)	R	(square root)	L	(natural log)
*	(multiplication)	S	(sine)	E	(e)

Parentheses are not permitted

COMMENTS: R,S,C,L and E operations are done first, all * and / operations next and all + and - operations last. Operations at the same level (e.g. + and -) are preformed in order from left to right. Mathematical errors casue a zero to be placed in the variable to the left of the equal sign. Values greater than 9,999,999,999 are not permitted. Rounding follows scientific convention (i.e. CALC I5=1.5. would set I5=2). If faster operation or rounding by truncation is desired, see FCALC and ICALC commands.

EXAMPLE: The CALC commands and tags listed in the left column are equivalent to the algebraic expressions or operations listed in the right column.

CALC	F22=I2+26-F7*6/42	$F22=I2+26-\frac{F7(6)}{42}$
CALC	F3=R36+L45*E4	$F3=\sqrt{36}+(\ln 45)(e^4)$
CALC	F8=SI7*SI7+CI5*CI5	$F8=\sin^2(I7)+\cos^2(I5)$
CALC	I23=F23	round the value of F23 to the nearest whole number.

COMMAND: CANT

TAG: A list of words separated by commas

OCCURRENCE: JUDGE-C

EFFECT AND COMMENTS: Use with MUST and DIDDLs (which see) in sentence judgers. If any word in the tag of a CANT appears in a student's response, it is overwritten with X's and the student's response is judged wrong. In addition, any WRITE, SHOW, JUMP, or other answer contingent commands immediately following the CANT are executed. The CANT command is provided so that you can base contingencies on particular foreseen errors. You may have several CANTs in a sentence judger, each with its own contingencies.

EXAMPLE:

```
.
.
.
UNIT    FARM18
WRITE   NAME SOME DOMESTIC FARM ANIMALS.
ARROW   510
MUST    COW,HORSE,CHICKEN,...
CANT    DEER,PHEASANT,...
WRITE   THIS IS A WILD ANIMAL
CANT    CORN,WHEAT,...
WRITE   THIS IS A PLANT, NOT AN ANIMAL.
.
.
.
```

(See also the DIDDL command)

COMMAND: CHAR

TAG: First line: a name for a special character (up to 7 letters long).

Following lines: up to 64 4-digit octal numbers separated by commas.

OCCURRENCE: Within any UNIT of a lesson (not necessarily the same UNIT in which it is used).

EFFECT: Allows a special character to be designed for a particular lesson.

COMMENTS: Special characters are displayed during the lesson by the PLOT command (which see). The CHAR command is used to design special characters which are to be used in addition to the characters in one of the standard character sets. Characters larger than the standard ones can be designed but you should seek expert advice before doing so.

The CHAR command specifies the points which are arranged within a standard area to form the shape of the desired special character. This area with its pattern of illuminated points can be positioned on the screen by a WHERE command.

Each point of a character is specified by a 4-digit octal number. The first 2 digits give a horizontal position and the second 2 digits give a vertical position. Horizontal positions for the standard characters range from 30 (left edge of character) to 36 (right edge of character). Vertical positions range from 44 (top) to 61 (bottom). Position 57 is just above the standard line when the "single number" WHERE command is used. Remember that these position numbers are expressed in octal notation, hence there are no positions 48, 49, 58, or 59.

EXAMPLE: This is how the presently used upper case "L" appears when written with a CHAR statement. The standard characters are, of course, directly available to the author through use of the WRITE command. In fact, Unit B3 and B4 below would look the same to a student. Remember that continuation lines are specified by a "blank" command.

```
UNIT    WHATSIS
CHAR    LCAP
        3044,3045,3046,3047,3050,3051,3052,3053
        3054,3055,3056,3656,3057,3157,3257,3357
        3457,3557,3657
.
.
UNIT    B3                        UNIT    B4
WHERE   201                       WHERE   201
PLOT    LCAP                      WRITE   L
```

COMMAND:	CLOCK
TAG:	TUTOR integer variable
OCCURRENCE:	UNIT-C, ARROW-C, ANS-C
EFFECT:	The time elapsed (in 60ths of a second) since student sign-in is placed in the TUTOR integer variable specified by the tag.
EXAMPLE:	Variable I12 has the elapsed time at which the student first saw the problem and variable I13 has the elapsed time at which he correctly solved the problem. The time in seconds spent on the problem is stored in I20.

```
UNIT     PROB2
WRITE    WHAT IS 2+2?
CLOCK    I12
ARROW    1020
ANS      4
CLOCK    I13
CALC     I20=I13/60-I12/60
WRONG
WHERE    1101
WRITE    WRONG,TRY AGAIN
```

COMMAND: DIDDL

TAG: A list of words separated by commas

OCCURRENCE: JUDGE-C

EFFECT and COMMENTS: Used with MUSTs and CANTs (which see) in sentence judgers. Any words in the tag of a DIDDL are permitted to be present in the student's response, but are not required to be so. A DIDDL with no tag, if given as the last command in a sentence judger, has the effect of permitting any words whatever (other than those in the tags of CANTs) to be present in the student's response. If a word appears in the student's response which is not a MUST word or a DIDDL word, it is overwritten with X's and the response is judged wrong.

EXAMPLE: The ANS command is included so PLATO has something to display if the student presses the ANS key. Note that the ANS-C for a correct answer appears after the first MUST command. UNIT EXTRA has a general DIDDL list which can be referenced from many different UNITs by using the JOIN command.

EXAMPLE:

```
UNIT    NURSE
WRITE   DIABETES IS A RESULT OF A MALFUNCTION
        IN THE...
ARROW   1001
ANS     ABILITY TO METABOLIZE SUGAR
MUST    METABOLISM, UTILIZATION, BURNING, TOLERATION,
        METABOLIZE, UTILIZE, USE, BURN, TOLERATE
WRITE   VERY GOOD
MUST    SUGAR, SUGARS, GLUCOSE, GLYCOGEN
CANT    FAT, FATS, PROTEIN, PROTEINS, VITAMIN,
        VITAMINS, CELLULOSE
WRITE   YOU MUST BE THINKING OF A DIFFERENT DISEASE
DIDDL   ABILITY, CAPABILITY
JOIN    EXTRA
.
.
.
END
UNIT    EXTRA
DIDDL   A,AFTER,AN,AND,ARE,AT,BEFORE,BY,
        CAN,DURING,FOR,FROM,IF,IN,INTO,
        IS,IT,MAY,OF,ON,OR,SHE,SHOULD,SINCE,
        THAN,THE,THEN,THERE,THROUGH,TO,TRY,
        USE,WHEN,WHILE,WITH
```

COMMAND:	END
TAG:	A blank tag is used
OCCURRENCE:	At end of a UNIT. Occurs in last UNIT of a main sequence program and in last UNIT of each *HELP* sequence.
EFFECT:	Causes "end of lesson" message to appear on the screen when the student tries to proceed from a main sequence UNIT followed by an END command. If UNIT is last of a HELP sequence, the student is returned to the main sequence when he tries to proceed.
COMMENT:	HELP and TERM units beyond the END command are accessable only by direct student request (via keyset) or JUMP type commands. If no END command appears in a lesson, PLATO acts as if there was one at the end of the last unit in the lesson. This command is useful for isolating UNITs which are used in branching operations from the main sequence UNITs.
EXAMPLE:	In this example the lesson ends after UNIT SP1-3. The HELP sequence for UNIT SP1-2 consists of UNITs HELP-3 and HELP-4. The END command in HELP-4 terminates that HELP sequence. The HELP sequence for SP1-3 consists of a single UNIT, HELP-5.

```
.
.
UNIT    SP1-1
.
.
UNIT    SP1-2
HELP    HELP-3
.
.
UNIT    SP1-3
HELP    HELP-5
.
.
END
UNIT    TERM-1
.
.
UNIT    HELP-3
.
.
UNIT    HELP-4
.
.
END
UNIT    HELP-5
.
.
END
.
.
```

COMMAND:	FCALC
TAG:	A TUTOR variable followed by an equal sign and an arithmetic expression. The arithmetic expression may consist of (a) a single constant or variable, or (b) two constants or variables separated by an operation symbol. All of the constants and TUTOR variables may be either integer or floating point. Indirect referencing (Chapter 8) is permitted.
OCCURRENCE:	UNIT-C, ARROW-C, ANS-C
EFFECT:	The value of the arithmetic expression is placed in the variable to the left of the equal sign. The only operation symbols allowed are + (for addition), - (for subtraction), * (for multiplication), and / (for division). When an integer variable is used on the left of the equal sign, fractional parts of the value of the arithmetic expression are ignored (e.g. FCALC I9=20.9 would set I9=20).
COMMENTS:	The FCALC command is performed faster than a comparable CALC command. Note that an FCALC command which contains only integer variables and constants could be replaced by a corresponding ICALC command, which would be more efficient and performed still faster. Integer constants must be less than 32,767.
EXAMPLE:	The following calculation sequence demonstrates several permitted types of FCALC expressions. Note that any fractional part of the expression F4+25.6 in the second FCALC command will be ignored.

```
UNIT      COMP4
FCALC     F5=10.3
FCALC     I6=F4+25.6
FCALC     F7=F10-I9
FCALC     F8=I6*F7
FCALC     F12=25/I8
FCALC     I14=75256.
```

The last FCALC indicates one proper procedure for cases where an integer constant greater than 32,767 is required.

COMMAND: GOTO

TAG: A UNIT name
Assigned Operations (Chapter 7) and Indirect Referencing (Chapter 8) are permitted.

OCCURRENCE: UNIT-C, ARROW-C, and ANS-C

EFFECT: Commands in the named UNIT are used to complete the current UNIT. The student remains in the current UNIT. Commands in the current UNIT which follow an executed GOTO command are never reached (unlike the JOIN command).

EXAMPLE: Both UNIT DEFINE and DEFINE2 do the same thing but DEFINE2 uses the Assigned Operation option.

```
UNIT    DEFINE                      UNIT    DEFINE2
WRITE   PRESS THE NUMBER            WRITE   PRESS THE NUMBER
        FOR THE WORD                        FOR THE WORD
        YOU WANT DEFINED                    YOU WANT DEFINED
        1  ALLELE                           1  ALLELE
        2  ALBINO                           2  ALBINO
        .                                   .
        .                                   .
        .                                   .
ARROW   1835                        ARROW   1835
LONG    1                           LONG    1
ANS     1                           STORE   12
GOTO    ALLELE                      ANS
.                                   GOTO    12,X,X,ALLELE,ALBINO,...
.                                   JUDGE   IGNORE
WRONG                               .
JUDGE   IGNORE                      .
.
.
UNIT    ALLELE
WHERE   1701
WRITE   ALLELE IS A TERM MEANING
```

The GOTO command is ideally suited for looping operations (see Chapter 11).

COMMAND: HELP (or HELP1, LAB, LAB1, DATA, DATA1)

TAG: A UNIT name. Assigned operations (Chapter 7) and indirect referencing (Chapter 8) are permitted.

OCCURRENCE and EFFECT: UNIT-C: Establishes a GENERAL HELP-type sequence for a UNIT. When the student presses the HELP key, he is branched to the UNIT mentioned in the HELP command tag. The student can then press NEXT to continue through the HELP sequence UNITS. When the student presses the NEXT key on the last HELP sequence UNIT or presses the SHIFT and BACK keys at any time in the HELP sequence, he will return to the MAIN UNIT from where he asked for HELP. This MAIN UNIT will appear as the student left it.

ARROW-C: Overrides any UNIT-C HELP-type sequence present.

ANS-C: The student is immediately jumped into the HELP-type sequence if his answer matches the ANSWER-TYPE command starting the ANS-C.

COMMENTS: Commands HELP1, LAB, LAB1, DATA, DATA1 all perform in the same way that HELP does except that a different key is used for each. HELP1, LAB1, or DATA1 commands are executed when both the SHIFT key and the HELP, LAB, or DATA keys are pressed. The last UNIT in each HELP-type sequence must have an END command as its last command.

EXAMPLE: UNIT SUMFAL1 is the GENERAL HELP for UNIT ECOL17. However, when the student is on ARROW 515, UNIT SPRING1 becomes the HELP. On ARROW 615 the student will immediately be sent to UNIT SUMFAL1 if he answers "PARROT".

```
UNIT    ECOL17
WRITE   NAME BIRDS THAT WOULD BE
        FOUND IN ILLINOIS WOODS IN
        THE INDICATED SEASON.

        SPRING -
        SUMMER -
HELP    SUMFAL1
ARROW   515
HELP    SPRING1
MUST    CARDINAL, BROWN THRASHER,...
ARROW   615
MUST    OVENBIRD, BLUEJAY,...
CANT    PARROT
HELP    SUMFAL1
   .
   .
   .
```

COMMAND:	ICALC
TAG:	A TUTOR integer variable followed by an equal sign and an arithmetic expression. The arithmetic expression may consist of either (a) a single integer constant or variable, or (b) two integer variables or constants separated by an operation symbol. Indirect referencing (Chapter 8) is permitted.
OCCURRENC:	UNIT-C, ARROW-C, ANS-C
EFFECT:	The value of the arithmetic expression is placed in the variable to the left of the equal sign. The only operation symbols allowed are + (for addition), - (for subtraction), * (for multiplication), and / (for division). The remainder in a division operation is ignored (e.g. 7/4 is interpreted as equal to 1).
COMMENTS:	The ICALC command is performed in less than 1/10 the time needed for a comparable CALC command (which see). Thus, in lessons which use extensive calculation routines, use of ICALC rather than the more flexible CALC command will result in fewer noticeable delays. Integer constants must be less than 32,767.
EXAMPLE:	The following calculation sequence demonstrates several permitted types of ICALC expressions. Note that any remainder in the division operation will be ignored.

```
UNIT    COMP3
ICALC   I6=10
ICALC   I5=I4+25
ICALC   I7=I10-I9
ICALC   I8=I6*I7
ICALC   I12=25/I8
.
.
.
```

COMMAND:	INFO
TAG:	A list of up to 10 TUTOR integer variables separated by commas.
OCCURRENCE:	UNIT-C, ARROW-C, ANS-C
EFFECT:	Whenever this command is encountered in a lesson (and collection of data on tape unit 4 has been requested), a record with the specified variables in placed on tape unit 4.
COMMENT:	Format of the record is similar to that of the standard student data record except that the word "INFO" appears instead of the response judgment (e.g. NO,SP) and a list of integer variables appears in the area where the student answer normally appears.
EXAMPLE:	I5 contains total correct answers while I6 contains total requests for Help during the preceding lesson. The universal wrong prevents the student from proceeding beyond this unit.

```
UNIT    ENDIT
WRITE   THIS IS THE END OF THE TEST
        HOW WELL DO YOU THINK YOU DID?
INFO    I5,I6
ARROW   401
LONG    300
WRONG
JUDGE   IGNORE
```

The following typical records might be produced on tape unit 4. Notice that values of the variables are right-justified in an 8-space field.

```
AVNER     26:33     ENDIT     1     INFO          25      126
AVNER     27:02     ENDIT     1     NO        REALLY  GREAT
```

COMMAND: INHIB

TAG: NEXT, NORING, ANSWER, ARROW, and/or OKNO

CONTINGENCY: UNIT-C: (NEXT and NORING tags only)

ARROW-C: (ANSWER, ARROW, and OKNO tags only)

EFFECT: Affects standard TUTOR student feedback options as follows

NEXT - The message -PRESS NEXT- will not appear on line 18 (usually appears as soon as the student has satisfactorily responded to all ARROWs in the UNIT)

NORING - Student keyset light will flash when student presses any key which is not acceptable in that UNIT (e.g., when the "HELP" key is pressed in a UNIT for which no HELP is provided). This light is not normally used for TUTOR lessons.

ANSWER - Student will not be able to receive the correct answer by pressing the "ANS" key (he normally can).

ARROW - An arrow will not be displayed at the position on the screen where a response is expected (it normally is).

OKNO - Messages OK, NO, SP, DP, etc. will not appear after response is judged correct, incorrect, misspelled, a duplicate, etc.

COMMENTS: These options are used for special effects where the standard feedback might be inappropriate. The ARROW-C options should be used cautiously since the feedback they inhibit is often useful to the author in finding errors in his lesson writing. It is probably best to limit use of the ARROW tag, for example, to UNITs which have only one ARROW.

EXAMPLES: This Unit accepts anything as an answer, hence the "ANS" key and the messages OK or NO are inappropriate. No -PRESS NEXT- message will appear at the bottom of the screen after the student has "judged" his entry by pressing Key "NEXT".

```
UNIT     WHO
WRITE    WHAT IS YOUR FIRST NAME?  PRESS KEY -NEXT-
         AFTER YOU HAVE TYPED IT.
INHIB    NEXT
ARROW    305
INHIB    OKNO,ANSWER
STORE    A3
ANS
.
.
.
```

COMMAND:	IPERM
TAG:	A single TUTOR integer variable or an integer constant. Indirect referencing (Chapter 8) is permitted.
OCCURRENCE:	UNIT-C, ARROW-C, ANS-C
EFFECT:	Fixes the upper bound to a set of integers from which subsequen selections are to be made at random without replacement. (The lower bound to the set is always on 1.) For example, IPERM 5 makes the set of integers 1,2,3,4 and 5 available for subsequent selection. (The actual selection is accomplished by means of th RANDP command, which see). The maximum value the tag may have i 96.
EXAMPLE	See RANDP.

COMMAND: JOIN

TAG: A UNIT name
Assigned Operations (Chapter 7) and Indirect Referencing (Chapter 8) are permitted

OCCURRENCE: Anywhere in a UNIT

EFFECT: Inserts the contents of the specified UNIT into the current UNIT. The student remains in the current UNIT.

COMMENTS: This command is useful when certain sequences of commands appear frequently in the same lesson. The sequence is written once and placed in a special UNIT which is "JOINed" to other UNITs as needed. ARROW commands must not appear within a UNIT which is JOINed to other UNITs. JOINed UNITs may also have JOIN commands within them. Such "nesting" of JOINed UNITs cannot be more than 6 deep.

EXAMPLE: UNIT TEST gives a student randomly drawn problems in addition, subtraction, and multiplication. The first two RANDU statements select numbers between 1 and 99 for use in the problem. The third RANDU statement selects a number between 1 and 3. This number is used by a later JOIN statement to set the rules - addition, subtraction, or multiplication - for the problem. The last JOIN statement attaches a perfect match judger to UNIT TEST.

```
UNIT     TEST
NEXT     TEST
JOIN     EXPLAIN
RANDU    I1,99
RANDU    I2,99
RANDU    I3,3
WHERE    911
SHOW     I1
WHERE    916
SHOW     I2
WHERE    920
JOIN     I3,X,X,ADD,SUB,MULT
WHERE    925
WRITE    =
ARROW    927
JOIN     JUDGER
```

```
UNIT     EXPLAIN
WRITE    DO THIS PROBLEM...
C
UNIT     ADD
CALC     I4=I1+I2
WRITE    +
C
UNIT     SUB
CALC     I4=I1-I2
WRITE    -
C
UNIT     MULT
CALC     14=I1XI2
WRITE    X
C
UNIT     JUDGER
STORE    15
ANS
CALC     I6=I5-I4
JUDGE    I6,NO,OK,NO
```

COMMAND: JUDGE

TAG: OK, NO, or IGNORE
Assigned Operations (Chapter 7) and Indirect Referencing (Chapter 8) are permitted.

OCCURRENCE: ANS-C (Must follow an answer-type command, e.g. ANS, WRONG, etc.)

EFFECT: Judging by the preceding answer-type command is overridden. New judging is based on the tag; (1) OK-(judge the response "OK"), (2) NO-(judge the reponse "NO"), (3) IGNORE-(erase the response and ignore it).

EXAMPLE: Both of the UNITS below ignore answers which are less than 1 or greater than 5. Answers 1, 3, or 5 are judged NO while 2 or 4 are judged OK. UNIT EVENT uses the assigned operation option.

```
UNIT    EVEN                            UNIT    EVENT
WRITE   TYPE AN EVEN NUMBER             WRITE   TYPE AN EVEN NUMBER
        BETWEEN 1 AND 5                         BETWEEN 1 AND 5
        THEN PRESS - NEXT                       THEN PRESS - NEXT
ARROW   520                             ARROW   520
ANS     2                               STORE   15
ANS     4                               ANS
WRONG   1                               JUDGE   15, IGNORE, IGNORE, NO,
WRONG   3                                       OK, NO, OK, NO, IGNORE
WRONG   5
WRONG
JUDGE   IGNORE
```

COMMAND:	JUMP
TAG:	A UNIT name Assigned Operations (Chapter 7) and Indirect Referencing (Chapter 8) are permitted.
OCCURRENCE:	UNIT-C, ARROW-C, and ANS-C
EFFECT:	Forces an immediate branch to the specified UNIT. The student automatically leaves the UNIT in which the JUMP command is encountered and enters the specified UNIT.
EXAMPLE:	Both UNITS perform the same function but MED-42 uses the assigned operation option.

```
UNIT    MED-41
WRITE   WHAT DATA DO YOU WANT?
        1) HER HEART RATE
        2) HER BLOOD PRESSURE
        3) HER TEMPERATURE
ARROW   320
ANS     1
JUMP    MED-78
ANS     2
JUMP    MED-94
ANS     3
JUMP    MED-87
WRONG
```

```
UNIT    MED-42
WRITE   WHAT DATA DO YOU WANT?
        1) HER HEART RATE
        2) HER BLOOD PRESSURE
        3) HER TEMPERATURE
ARROW   320
STORE   133
WRONG
JUMP    133,X,X,MED-78,MED-94,
        MED-87,X
```

COMMAND: LINE

TAG: 1. Two 4-digit numbers (separated by a comma) which specify 2 points on the screen. For each 4-digit number, the first two digits specify one of 18 lines (01 is the top line). The second two digits specify a character position on that line (01 is the left-most space). If the first digit of the tag is zero it may be omitted (i.e., 101 is the same as 0101.). TUTOR integer variables are also permitted as well as indirect referencing (see Chapter 8).

2. Four numbers separated by commas. For more precise positioning a "four number" LINE command is used. These numbers represent, respectively, the starting Y-value, the starting X-value, the ending Y-value, and the ending X-value. See the description of the "double-number" WHERE tag for details of this finer scale coordinate system.

OCCURRENCE: UNIT-C, ARROW-C, ANS-C.

COMMENT: End points of lines appear where the center of a character would be if that character was positioned by a WHERE command having a tag with the coordinates of the end-point.

EXAMPLE: In this example a line is drawn under the space in which the student's response will appear.

```
.
.
.
UNIT    SP1-1
WHERE   801
WRITE   TYPE YOUR NAME HERE
LINE    1021,1040
ARROW   920
ANS
JUMP    SP1-2
.
.
.
```

COMMAND: LONG

TAG: A number specifying the maximum number of characters in a student's response.

OCCURRENCE: JUDGE-C

EFFECT: When a student response reaches the length indicated in the tag the computer will automatically check the response.

COMMENTS: If a LONG command is NOT included, the computer assumes a LONG with the tag of 32. The student may request the computer to check his response by pressing the "NEXT" key at any time before the response length exceeds the length specified in the tag of the LONG command.

EXAMPLE: In the example below the student's response on the first ARROW is evaluated as soon as the first letter is entered. The second ARROW allows for up to 2 lines (96 letters) of student response.

```
UNIT          95
WRITE         HAVE YOU STOPPED BEATING YOUR WIFE?
              ANSWER YES OR NO
              COMMENT ON THIS QUESTION
              (USE 2 LINES OR LESS)
ARROW         218
LONG          1
ANS           Y
WRITE         GOOD, EVERYONE HAS BEEN TALKING ABOUT YOU
ANS           N
WRITE         THATS OUTRAGEOUS
ARROW         501
LONG          [illegible]
ANS
  .
  .
  .
```

COMMAND:	LOOP
TAG:	A number between 1,000 and 10,000,000.
OCCURRENCE:	Before a group of commands which will be repeatedly activated
EFFECT and COMMENTS:	A "fatal loop" occurs when the same set of commands is executed over and over again without end. To prevent fatal loops from occurring, no more than 1000 TUTOR commands are processed for a given contingency. Exceeding this limit will cause the student to be removed from the lesson and an error message to be placed on his screen. Since most contingencies seldom exceed 10 commands, accidental "fatal loops" are quite effectively caught without hindering the usual author. However, some programming (e.g. graphing) requires looping which causes several thousand commands to be activated. The LOOP command is available to those authors who need to override the 1000 command limit.

CAUTION:

1. Do not use LOOP unless you know what you are doing
2. Do not use LOOP unless it is actually needed (i.e., you have gotten a "FATAL LOOP" error message).
3. Do not place the LOOP command within the loop or use more than one LOOP command within the same contingency.
4. The number in the tag should only slightly exceed the number of commands you expect to process.
5. The LOOP command must be located within the contingency which requires its use.

COMMAND: MOVE

TAG: 4 arguments separated by commas; 1 and 3 are TUTOR variables (integer or word format), 2 and 4 are integers or TUTOR integer variables which specify character positions in the variables given in the first and third arguments respectively. Indirect referencing is permitted.

EFFECT: Allows movement of a character from one TUTOR variable to another. The specified character in the first TUTOR variable replaces the specified character in the second TUTOR variable. The character in the first TUTOR variable is unchanged.

COMMENTS: A "character" is a 6-bit piece of a 48-bit TUTOR variable. The 8 characters making up a TUTOR variable are numbered from 1 to 8 (left to right). Specification of a character position greater than 8 is interpreted as a referenco to the appropriate position in a following variable. Thus, a reference to position 9 of variable 60 is interpreted as a reference to position 1 of variable 61. Attempts to reference variables greater than 63 will produce an error message. Remember that while character positions for "A" format variables correspond to the 8 character positions, the same correspondence does not obtain between character positions in "I" format variables and decimal digit positions.

EXAMPLE: The lesson below is a 40-question multiple-choice exam. The questions are on slides 1-40. The MOVE command in Unit CHOICE is used to store the student's answers packed 8 per variable starting in variable 31. The MOVE commands in Unit COUNT are used to unpack the student's answers and the author's answers. For each match, counter I2 is incremented by 1.

```
UNIT    PRE
C       THE CORRECT ANSWERS
ACALC   A41=ABACDEEA
ACALC   A42=CCDABECD
ACALC   A43=CAADEAAB
ACALC   A44=BACEEACB
ACALC   A45=BBDEAACD
C       TOTAL NUMBER OF PROBLEMS
CALC    I9=40
CALC    I1=1
JUMP    CHOICE
C
UNIT    CHOICE
SLIDE   I1
ARROW   1010
INHIB   OKNO
LONG    1
STORE   A3
ANS
MOVE    A3,1,A31,I1
ADD1    I1
CALC    I2=I9-I1
JUMP    I2,TALLY,CHOICE
```

```
UNIT    TALLY
NEXT    TALLY
WRITE   YOUR SCORE IS     PER CENT.
ZERO    I2
ZERO    I3
ZERO    I4
CALC    I1=I9
JOIN    COUNT
CALC    I4=I2/I9*100
WHERE   110
SHOW    I4
C
UNIT    COUNT
MOVE    A31,I1,I3,8
MOVE    A41,I1,I4,8
CALC    I5=I3-I4
GOTO    I5,COUNT1,X,COUNT1
ADD1    I2
GOTO    COUNT1
C
UNIT    COUNT1
SUB1    I1
GOTO    I1,X,X,COUNT
```

COMMAND: MUST

TAG: A list of words separated by commas.

OCCURRENCE: JUDGE-C

EFFECT AND COMMENTS: A set of MUST, CANT, and DIDDL commands associated with an arrow comprise a "sentence judger' for that arrow. The words in the tag of a MUST are treated as interchangeable synonyms, and at least one of them must be present in the student's response. If several MUST commands are listed together, then at least one word from each of them must be present in the student's response. Words not required by MUST commands may be present in the student's response only if they are explicitly permitted. Concurrent use of the SPELL command will cause PLATO to check the student's answer for misspellings of MUST words. Misspellings are then underlined in the student's answer and must be corrected before the answer is judged "OK." Spaces, commas, periods, and question marks are used to separate words in the student's sentence and thus must not be used as part of a word in a sentence. Answer contingencies for correct answers should be positioned after the first MUST command.

EXAMPLE:

```
UNIT    NUMS8
WRITE   WHAT NUMBERS ARE BETWEEN 3 and 6?
ARROW   510
MUST    4,FOUR
WHERE   601
WRITE   VERY GOOD
MUST    5,FIVE
DIDDL   AND,+
```

COMMAND: NEXT

TAG: A UNIT name
Assigned Operations (Chapter 7) and Indirect Referencing (Chapter 8) are permitted

OCCURRENCE:

UNIT-C: Specifies the next unit of study

ANS-C: Causes "NEXT" key to be only "legal" key and specifies which UNIT will be obtained when that key is pressed. This is a nice WRONG answer contingency for branching since it allows the student to see his error before going to the next UNIT.

COMMENTS: In the absence of a NEXT UNIT-C command, the UNIT following the current UNIT will be obtained by the student when all his answers are "OK" and he pushes the NEXT key.

EXAMPLE: In this example NEXT commands are used as UNIT and answer-type contingencies. If the student is wrong, for example, he is sent back to UNIT SP1-3. If he is correct, he goes to UNIT SP1-10 when he presses key NEXT.

```
UNIT    SP1-9
NEXT    SP1-10
WHERE   818
WRITE   3+3=
ARROW   822
ANS     6
ANS     SIX
WRONG
NEXT    SP1-8
.
.
.
```

COMMAND:	NODUP
TAG:	A number or a TUTOR integer variable which ranges from 1 to 63. Indirect referencing (Chapter 8) is permitted.
OCCURRENCE:	Ans-C
EFFECT:	Authors often ask questions of the type, "list three reasons for....". The order in which the student lists the answers is immaterial. However, the student is not to be allowed to list the same response more than once- a condition called "duplicate answers." The NODUP command following a matched ans-type statement directs the computer to check if any other judged answer in the Unit had a similar NODUP statement tag. If so, the answer is judged a <u>duplicate answer</u> and DP is placed after the answer on the student's screen. The student must erase this answer.
EXAMPLE:	In this example, the student cannot have duplicate answers.

```
.
.
.
UNIT      PLANETS
WRITE     LIST THREE PLANETS OF OUR SOLAR SYSTEM.
ARROW     510
JOIN      9PLAN
ARROW     610
JOIN      9PLAN
ARROW     710
JOIN      9PLAN
C
UNIT      9PLAN
ANS       MERCURY
NODUP     1
ANS       VENUS
NODUP     2
ANS       EARTH
NODUP     3
ANS       OUR PLANET
NODUP     3
ANS       MARS
NODUP     4
ANS       JUPITER
NODUP     5
ANS       SATURN
NODUP     6
ANS       URANUS
NODUP     7
ANS       NEPTUNE
NODUP     8
ANS       PLUTO
NODUP     9
```

COMMAND: PLOT

TAG: The name of a special character which is defined within the lesson by a CHAR command.

OCCURRENCE:

UNIT-C: special character appears on student's screen whenever the student is in the UNIT.

ARROW-C: special character appears on screen when the student is working on the particular arrow. Disappears when the student works on a different arrow.

ANSWER-C: special character appears when the student's response matches an Answer-type command.

COMMENTS: The special character is generally positioned by a WHERE command which preceeds it. See comments for WRITE and SLIDE also.

EXAMPLE: The special character name "POINTER" (which is defined elsewhere in the same lesson by a CHAR command) is positioned on the screen by a WHERE command. It is used in the example to cause a small arrow to be pointed at different areas on slide 25 which has a map of Illinois on it.

```
UNIT    MAP
SLIDE   25
WRITE   TYPE THE NAME OF THE CITY WHOSE LOCATION
        IS DESIRED. THEN PRESS KEY -NEXT-. AN
        ARROW WILL POINT TO THE CITY ON THE MAP.
ARROW   1820
ANS     URBANA
WHERE   1235
PLOT    POINTER
ANS     PAXTON
WHERE   1034
PLOT    POINTER
.
.
.
```

COMMAND: PUT

TAG: Two character strings of the same length separated by an equal sign.

OCCURRENCE: JUDGE-C

EFFECT: Any character in the student's response identical to a character in the first string of characters of the PUT command tag is changed for judging to the character in the identical position in the string after the equal sign.

COMMENTS: In certain fields (e.g., genetics) the student's response will often be in symbols. These symbols may have characters that vary from problem to problem. However, an underlying logic exists for symbol construction in all the problems. Thus, it is advantageous to write a general symbol judger that can be added to many UNITs by use of the JOIN command. All one need do is PUT the particular characters for a problem equal to the characters in the general symbol judger.

EXAMPLE:

```
UNIT    GENET8
WRITE   WHAT IS THE PROBABLE GENOTYPE OF A
        NORMAL MOTHER WHO HAS AN ALBINO CHILD?
ARROW   410
ANS     +A
PUT     A=M
JOIN    GCHECK
.
.
.
END
UNIT    GCHECK
WRONG   ++
WRONG   +M
WRONG   MM
WRONG   M+
WRITE   AS A CUSTOM, + ALWAYS PRECEDES
        THE MUTANT GENE SYMBOL
WRONG   + +
WRITE   DO NOT PUT SPACES IN GENOTYPES
WRONG   + M
WRITE   DO NOT PUT SPACES IN GENOTYPES
WRONG   M M
WRITE   DO NOT PUT SPACES IN GENOTYPES
BUMP    + M
WRONG
WRITE   THE GENOTYPE IS INCORRECTLY WRITTEN
WRONG
WRITE   PERHAPS YOU ARE USING THE WRONG GENES
```

COMMAND: RANDP

TAG: A single TUTOR integer variable. Indirect referencing (Chapter 8) is permitted.

OCCURRENCE: UNIT-C, ARROW-C, ANS-C

EFFECT: Randomly selects an integer from a set of integers provided by an IPERM command and puts it into the variable named in the tag. The integer selected is then eliminated from the set and the next selection is made from among those remaining. When the set has been exhausted, RANDP will "select" zero.

COMMENTS: It is possible at any time to abandon one set of integers and start selecting from another by executing a new IPERM command.

EXAMPLE

```
.
.
UNIT    BEGIN
WRITE   PRESS NEXT TO BEGIN
        THE DRILL.
IPERM   30
.
.
UNIT    BRANCH
RANDP   I8
JUMP    I8,DONE,DR1,DR2,..,DR30
.
.
UNIT    DR1
NEXT    BRANCH
WRITE    37
        +78
LINE    301,304
ARROW   401
ANS     115
.
.
```

```
UNIT    DR30
NEXT    BRANCH
WRITE    13
        -71
LINE    301,304
ARROW   401
ANS     -58
.
.
UNIT    DONE
WRITE   YOU HAVE FINISHED
        THE DRILL
END
.
.
```

COMMAND: RANDU

TAG: Either

1. A single TUTOR floating point variable or
2. A TUTOR integer variable followed by a comma and another integer variable or constant.

Indirect referencing (Chapter 8) is permitted.

OCCURRENCE: UNIT-C, ARROW-C, ANS-C

EFFECT:

1. In the floating point option, a pseudorandomly generated floating point number between zero and one is returned in the variable mentioned in the tag.

2. In the integer option, a pseudorandomly generated integer number between 1 and the value of the second variable or constant is returned in the first variable.

COMMENTS: A collection of the pseudorandomly generated numbers produces a uniform distribution of points within the limits of the number range.

EXAMPLE:

```
.
.
UNIT    RANDOMF
WRITE   HERE IS A RANDOMLY GENERATED
        NUMBER BETWEEN ZERO AND ONE.
        A COLLECTION OF THESE NUMBERS
        WILL YIELD A UNIFORM DENSITY
        BETWEEN 0 AND 1.
RANDU   F31
WHERE   810
SHOW    F31
.
.
UNIT    RANDOMI
WRITE   HERE IS A RANDOMLY GNERATED
        INTEGER.  A COLLECTION OF
        THESE INTEGERS WILL PRODUCE A
        UNIFORM DENSITY BETWEEN 1 AND 100.
RANDU   I3,100
WHERE   810
SHOW    I3
.
.
.
```

COMMAND: RESET

TAG: None

OCCURRENCE: JUDGE-C

EFFECT: Restarts judging.

COMMENTS: At the start of a JUDGE-C, a copy of the student's response is obtained. During the JUDGE-C, many changes may occur in this copy. A BUMP command will eliminate letters from the copy, a PUT command will replace letters, and sentence judging will cause a complete restructuring of the copy. An author may want to try several types of judging successively on a simple answe. A "fresh" copy of the student's response must be obtained before each type of judging is tried. The RESET command does just this. If a student's response is not matched by any of the answer type commands before a RESET command occurs, a new copy of the student's response is obtained and judging starts afresh in accordance with commands that follow the RESET command.

EXAMPLE: In the example below, the RESET command allows judging of many different possible sentence answers.

```
UNIT     SOUTH
WRITE
         NAME A SOUTHERN STATE ALONG
         WITH THE CAPITAL OF THAT STATE--
ARROW    1001
MUST     GEORGIA
MUST     ATLANTA
JOIN     EXTRAS
RESET
MUST     FLORIDA
MUST     TALLAHASSEE
JOIN     EXTRAS
RESET
MUST     ALABAMA
MUST     MONTGOMERY
JOIN     EXTRAS
RESET
.
.
.
UNIT     EXTRAS
DIDDL    STATE,CAPITAL,IN,THE,..etc.
```

COMMAND: SHOW

TAG: A single TUTOR alphabetic, integer, or floating point variable. Indirect referencing (Chapter 8) is permitted.

OCCURRENCE:

UNIT-C: The contents of the variable named appear on the student's screen whenever the student is working on the UNIT.

ARROW-C: The contents of the variable named appear on the screen when the student is working on the given arrow.

ANS-C: The contents of the variable named appear when in judging the compute. matches the answer-type command starting the ANS-C.

COMMENTS: Generally, a SHOW command should be preceded by a WHERE command stating the location on the screen for the SHOW display. The display formats for the SHOW command are A8, I8, or F16.7. See STORE command comments.

EXAMPLE: In this example six blank spaces precede the integer and floating variable and two blank spaces follow the alphabetic variable. Other printing (by WRITE, SHOW or PLOT commands) could be placed where these blanks now appear. Note that seven digits always follow the decimal point of a floating point variable. Variable A2 contains the word "TWENTY", I4 contains the number "20" and F7 contains the number "20.0".

```
UNIT    SEE
WHERE   201
SHOW    A2
WHERE   301     is seen on the screen as     TWENTY  20
SHOW    I4                                         20.0000000
WHERE   401
SHOW    F7
```

COMMAND: SLIDE

TAG: A number or a TUTOR integer variable which ranges from 0 to 122 to specify a slide. Indirect referencing (Chapter 8) is permitted.

OCCURRENCE: UNIT-C, ARROW-C, ANS-C

COMMENTS: Slides can be presented on the screen much more rapidly than plotting from WRITE or PLOT commands can. Thus, where presentations involve rapid sequences of different displays or displays with extensive amounts of material on them, slides will produce better results than WRITE commands. Diagrams are most elegantly done by use of slides. WRITE, PLOT and SLIDE commands can be used together to provide superimposed images for special effects.

EXAMPLE: In this example slide number 4 is shown with the question "Which slide is this?" superimposed on it by use of a WRITE command.

```
.
.
.
UNIT    SP1-3
SLIDE   4
WRITE   WHICH SLIDE IS THIS?
ARROW   0201
ANS     4
.
.
.
```

COMMAND:	SPELL
TAG:	None
OCCURRENCE:	JUDGE-C
EFFECT:	The computer checks the student's response for possible misspellings of the author's ANS or MUST command tag. SP is placed after the student's answer if it is a misspelling of the author's ANS tag. In sentence judging, if one of the author's MUST words is misspelled by the student, that word in the student's answer is underlined. The student cannot continue until his misspellings are corrected.
COMMENTS:	Deletions and insertions are handled in addition to incorrect characters.

EXAMPLE:

```
UNIT    GEOM16
WRITE   WHAT IS THE NAME OF A
        TRIANGLE HAVING TWO EQUAL
        SIDES?
ARROW   410
SPELL
ANS     ISOSCELES
.
.
.
```

Any of the following student answers will be interpreted as misspellings of the Answer by the computer:

```
isocles       isocales       isasales
isoscles      isosales       esasceles
isasceles     icoseles       asoseles
```

COMMAND: STORA

TAG: A single TUTOR integer or floating point variable.

OCCURRENCE: JUDGE-C

EFFECT: An arithmetic student response is evaluated and stored in the variable designated in the tag. The permitted operations are addition (+), subtraction (-), multiplication (*), division (÷), square root (R), sine (S), cosine (C), natural logarithm (L), and raising e to a power (E). Any error returns a zero in the variable.

COMMENTS: STORA allows a student terminal to be used as a simple desk calculator. The comments for the CALC command also apply for STORA.

EXAMPLE:

```
        .
        .
        UNIT     ARITH
        WRITE    THIS UNIT IS A CALCULATION UNIT.

                 WRITE THE ARITHMETIC EXPRESSION
                 YOU WISH EVALUATED.  THEN PUSH -NEXT-.
                 USE
                 + FOR ADDITION                C FOR COSINE
                 - FOR SUBTRACTION             S FOR SINE
                 * OR x FOR MULTIPLICATION     L FOR NATURAL LOG
                 / OR ÷ FOR DIVISION           E FOR E TO A POWER
                 R FOR SQUARE ROOT
        ARROW    1301
        STORA    F35
        STORA    I21
        ANS
        WHERE    1501
        WRITE    THE VALUE OF THE EXPRESSION YOU HAVE
                 WRITTEN IS...
                 OR AS AN INTEGER...
        WHERE    1613
        SHOW     F35
        WHERE    1720
        SHOW     I21
        .
        .
        .
```

COMMAND: STORE

TAG: A single *TUTOR* alphabetic, integer, or floating point variable. Indirect referencing (Chapter 8) is permitted.

OCCURRENCE: JUDGE-C

EFFECT: When the computer judges a given arrow, the student's answer is stored in the variable named in the tag.

COMMENTS: The student's answer is stored in the format given by the author: A for alphabetic string, I for integer, and F for floating point. Any non-numeric characters are ignored while storing under integer and floating point formats.

Since only 8 characters can be stored under A format in a variable, consecutive A format STORE commands are designed to store consecutive blocks of 8 characters of the student's answer.

EXAMPLE:

```
UNIT   LANG
WRITE  TRANSLATE THE SENTENCE
       'WIE GEHT ES IHNEN?'
ARROW  301
STORE  A1
STORE  A2
STORE  A3
ANS
.
.
.
UNIT  TRANS
WRITE HERE IS YOUR LAST TRANSLATION
      OF 'WIE GEHT ES IHNEN?' YOU
      MAY GIVE A DIFFERENT TRANSLATION,
      OR TYPE -N- FOR NO CHANGE.
WHERE 601
SHOW  A1
SHOW  A2
SHOW  A3
.
.
ANS   N
JUMP  OK
ANS
```

COMMAND: SUB1

TAG: A single TUTOR integer variable. Indirect referencing (Chapter 8) is permitted.

OCCURRENCE: UNIT-C, ARROW-C, ANS-C.

EFFECT: Decreases by one the value of the variable listed in the tag.

EXAMPLE: Variable I5 is used here to keep track of the total number of correct answers minus total wrong answers for a lesson. A correct answer increases I5 by one, an incorrect answer decreases I5 by one.

```
.
.
UNIT    PROB1
WRITE   WHAT IS...
ARROW   1010
ANS     ..
ADD1    I5
WRONG
SUB1    I5
.
.
.
.
```

COMMAND: TERM

TAG: A single word

OCCURRENCE: Anywhere within the UNIT.

EFFECT: If the student anywhere in the lesson presses the TERM key and writes out a word identical to the tag of a TERM command, he will immediately branch to the UNIT containing this TERM command. When the student presses the NEXT or BACK key in this new UNIT, he will return to the previous UNIT he was working on.

COMMENTS: Many TERM commands may occur in a single UNIT. The only change adding a TERM command to a UNIT produces is that the student has access to this UNIT from anywhere. This command is ideally suited for word definitions, reference tables, review, etc.

More exactly, only the first eight characters of the student's and author's terms are examined for matching.

EXAMPLE:

```
UNIT    HIST54
TERM    ROMULUS
TERM    REMUS
WRITE   ROMULUS AND REMUS WERE
        LEGENDARY TWIN BROTHERS
        WHO WERE RAISED BY A
        SHE-WOLF....
.
.
.
```

UNIT HIST54 is now available to the student anywhere in the lesson when he asks for the term Romulus or Remus.

COMMAND: TIME

TAG: A number or a TUTOR integer variable whose value represents a time duration expressed in units of 1/60 second. Indirect referencing (Chapter 8) is permitted).

OCCURRENCE: UNIT-C, ARROW-C, ANS-C

EFFECT: Presses the "NEXT" key for the student after the given time has elapsed from beginning of the contingencies.

COMMENTS: The TIME command should be the last of the contingencies.

EXAMPLE: In this example the student has 5 seconds (300 60ths of a second) to give an answer to the problem. If he gives the correct answer he is immediately shifted to UNIT TEST6. If he gives the wrong answer he is shifted to UNIT WRONG (The contingency of the "universal WRONG"). If he makes no response (a "blank input") he is shifted to UNIT BLANK (the contingency of the first WRONG command with a blank tag). Note that only the last WRONG with a blank tag acts as a "universal WRONG" if more than one is present.

```
UNIT    TEST5
WRITE   2+2=
TIME    300
ARROW   106
LONG    1
ANS     4
JUMP    TEST6
WRONG
JUMP    BLANK
WRONG
JUMP    WRONG
.
.
.
UNIT    BLANK
WRITE   SORRY, YOUR TIME RAN OUT...
.
.
.
UNIT    WRONG
WRITE   SORRY, YOUR ANSWER WAS WRONG...
.
.
.
```

COMMAND:	T6ØTHS
TAG:	NONE
OCCURRENCE:	In first UNIT of a lesson
EFFECT:	This command is used to alter the usual time on the student data kept on magnetic tape. Normally the time of occurrence of each data record is indicated in minutes and seconds (where zero minutes and zero seconds represents the time that the student started the lesson). In cases where more precise timing information is desired the T6ØTHS command gives time in 6Øths of a second.
EXAMPLE:	Each of the lines below (except the heading) represents the same student record for a student answer ("ILLINOIS") given 3 minutes and 4 seconds after the lesson began. The first record is the usual form of student record while the second shows how 3 minutes and 4 seconds (11,14Ø 6Øths of a second) would be shown in a lesson having the command T6ØTHS in its first UNIT.

NAME	TIME	UNIT	ARROW	JUDGE	STUDENT ANSWER
AVNER	3:Ø4	PØ4	1	NO	ILLINOIS
AVNER	1114Ø	PØ4	1	NO	ILLINOIS

COMMAND:	UNIT
TAG:	A name (the "UNIT name") up to 7 characters in length. No spaces or commas should appear in name.
OCCURRENCE:	First command in each UNIT (the online editing does this automatically for you).
EFFECT:	Begins the basic addressable unit in the lesson (e.g., a screenful of information or a distinct computational routine). Ends previous UNIT (if any). Initiates UNIT contingency.
COMMENTS:	The "UNIT name" is used to aid the computer in the interconnection of UNITS, in the selection of a UNIT for on-line editing and to indicate the location of the student in the lesson on the student data. These names must be unique and should be chosen for ease in use.
EXAMPLE:	The first three UNITs in the first lesson in a Sample Program might be names as follows:

```
UNIT   SP1-1
.
.
.
UNIT   SP1-2
.
.
.
UNIT   SP1-3
.
.
.
```

COMMAND:	UPLOW
TAG:	No tag is used
OCCURRENCE:	In first UNIT of a lesson
EFFECT:	This command makes the standard upper case, lower case and Cyrillic character sets available for the lesson. It must be present if the ANSRU and WRUSS commands are to be used in a lesson.
EXAMPLE:	UNIT BLURB1 is the first UNIT in a lesson using upper and lower case characters.

```
UNIT     BLURB1
AREA     GRAPHIC DISPLAYS IN ADVERTISING
UPLOW
WRITE    This lesson will demonstrate...
.
.
.
```

COMMAND: WHERE

TAG: Two types of tags are accepted, (1) a single number (or TUTOR integer variable) or, (2) two numbers (or TUTOR integer variables) separated by a comma. Indirect referencing (Chapter 8) is permitted.

(1) To position writing on the screen, a single 4-digit number is usually used. The first two digits specify one of the 18 lines (01 is the top line). The second two digits specify a character position on that line and can range from 01 (the first character position) to 48 (the last character position). If the first digit of the tag is zero it may be omitted (i.e., 101 is the same as 0101).

(2) When finer control of location is desired, a tag consisting of two numbers (separated by a comma) can be used. The first number specifies one of 170 vertical positions (0 is the top position). The second number specifies one of 240 horizontal positions (0 is the left of the screen).

OCCURRENCE: Just before a WRITE, SHOW, or PLOT command.

EFFECT: Specifies the screen position where display of the tag of the following WRITE, SHOW, or PLOT command is to begin.

COMMENTS: In terms of the second type of tag the standard characters used in TUTOR are written within a rectangle 10 units high and 5 units wide. The "double number" tag refers to the upper-left corner of this rectangle.

EXAMPLES: The letter X is displayed on line 9, space 4 (upper case character set), by all of the following Units.

```
UNIT    A       UNIT    C        UNIT    E        UNIT    G
WHERE   0904    CALC    I1=904   CALC    I1=80    CALC    I2=15
WRITE   X       WHERE   I1       CALC    I2=15    WHERE   80,I2
.               WRITE   X        WHERE   I1,I2    WRITE   X
.               .                WRITE   X        .
.               .                .                .
UNIT    B       .                .                .
WHERE   904     UNIT    D        .
WRITE   X       WHERE   80,15    UNIT    F
.               WRITE   X        CACL    I1=80
.               .                WHERE   I1,15
.               .                WRITE   X
                .                .
                                 .
                                 .
```

COMMAND: WRITE

TAG: A message with up to 18 lines of 48 characters each.

OCCURRENCE:

UNIT-C: tag appears on the student's screen whenever the student is in the UNIT.

ARROW-C: tag appears on screen when the student is working on the particular arrow. Disappears when the student works on a different arrow.

ANSWER-C: tag appears on screen when student's response matches an Answer-type command.

COMMENTS: Unless overridden by a WHERE command, the writing of the first WRITE command begins in the first space of the first line during a UNIT contingency and in the first space of the 17th line during an ANSWER contingency. Arrow contingency writing should always be preceded by a WHERE command. Avoid writing on the 18th line since it is used for operational messages by the computer (e.g., "PUSH NEXT TO CONTINUE"). See comments for SLIDE also. The command WRUSS is used to produce Cyrillic characters when the UPLOW character set is available. If several display statements (WRITE, SHOW, etc.) appear as an ARROW-C or ANS-C, only the last such statement will be erased when a new ARROW-C is activated or the answer is erased.

EXAMPLE: The message in the tag of the WRITE command is displayed on the student's screen. The second line has been indented by using "space" characters - just as on a typewriter.

```
UNIT    HI
WHERE   801
WRITE   WELCOME TO PLATO
            PRESS NEXT TO CONTINUE
.
.
.
```

COMMAND: WRONG

TAG: An expected wrong answer

OCCURRENCE: JUDGE-C

EFFECT: If the student's response matches the tag of a WRONG command a "NO" is placed on the screen after his response. Any Answer-type contingencies following the WRONG command are then initiated.

COMMENTS: Any number of WRONG commands can be used after the same ARROW. A WRONG command with a blank tag would be matched only when the student's response is blank unless this WRONG command is the last ANS-type command for ARROW. When the later case exists, the command is known as a "UNIVERSAL WRONG" and will cause any answer that does not match the tag of another WRONG command or an ANS command to be scored "NO". Even if no WRONG command follows an ARROW command, the computer will act as if a WRONG with a blank tag was the last Answer-type command before the next ARROW, UNIT or END command. A WRGRU command should be used when the student is using Cyrillic characters.

EXAMPLE: In this UNIT the first WRONG with a blank tag is matched only if the student's response was blank (i.e. he asked PLATO to "judge" his answer before he gave one). The second WRONG with a blank tag is a "UNIVERSAL WRONG."

```
.
.
UNIT      SP1-7
          WHAT COLOR IS A BLACKBOARD?
ARROW     226
ANS       BLACK
WRONG     GREEN
WRITE     I WOULD HAVE SAID "GREENBOARD" THEN
WRONG
WRITE     YOU DID NOT ANSWER YET.
WRONG
WRITE     ARE YOU TRYING TO BE FUNNY?
.
.
.
```

COMMAND: ZERO

TAG: A single TUTOR integer variable. Indirect referencing (Chapter 8) is permitted.

OCCURRENCE: UNIT-C, ARROW-C, ANS-C.

EFFECT: Sets to zero the value of the variable listed in the tag.

COMMENTS: This command is used to initialize a counter.

EXAMPLE:

```
.
.
UNIT    BEGIN
ZERO    I5
ZERO    I6
WRITE   THIS LESSON WILL CONSIST
        OF 35 PROBLEMS.
.
.
.
```

Counters I5 and I6 could now be used to keep track of correct and wrong answers by using the ADD1 command.

Chapter 6

VARIABLES

Chapter 5 briefly touched on the use of TUTOR variables. It was mentioned there that TUTOR provides for 63 information "storage spaces" for each student. These spaces, or "TUTOR variables" may be used to store information with three different types of format: (A) alphabetic - words, letters, symbols, etc.; (I) integers - 1,73,0, etc.; and (F) fractional numbers - 1.0, 27.428, etc. Unlike the lesson structure, TUTOR variables can be altered during a lesson, hence the use of the term "variable". For example, UTOR variable "15" in "I" format (more simply referred to as "variable I15") might be used to individualize the lesson by giving certain portions of the lesson material only to students whose I15 had a value greater than 25. Variable I15 might contain a count of the number of correct or incorrect responses made by the student during a short test given by PLATO earlier in the same lesson. Another use of TUTOR variables would be to serve as a place to put numbers or words temporarily during a lesson. PLATO could then, for example, show the student the answer to a previous problem needed for the solution of another problem. The same TUTOR variable might be used again and again during a single lesson for several different purposes. The format of any TUTOR variable can be altered within the lesson and the author may set the variable to any desired value or allow PLATO or the student to define or alter the value.

Format

New authors may wonder why it is necessary to specify the format of a TUTOR variable. A human can tell that "cat" is a word, "25" is an integer, and "15.32" contains a decimal fraction simply by looking at the expression itself. However, PLATO stores its information completely in the form of numbers and the "format" code letter tells PLATO how to decode the numbers to return them to their original form when needed.

A simple example may help. Imagine that you have a dial which can be set to any number from 0000 to 9999. Obviously you could use this dial to remember any number you might want to keep track of (like your wife's age) simply by setting it to that number. This use of the dial is analogous to a TUTOR variable in I (interger) format. Whenever your wife had a birthday you could reset the dial to the next higher number. This operation is just like the use of the TUTOR command "ADD1" which adds the number "1" to a TUTOR integer variable. By a simple code you can extend your dial to a memory device for symbols. You could define 01=A, 02=B, and so forth up to 26=Z. By adding various numerals, punctuation marks, arithmetic symbols, etc. you could easily use up 99 numbers in such a code. Since your dial goes up to 9999 you could use it to remember 2 symbols by letting the first 2 digits on your dial be the code for the first symbol and the second 2 digits be the code for the second symbol. Thus 0102 would be divided into 01=A and 02=B or "AB". If "30" was the code for the numeral "4" and "32" was the code for the numeral "6" you could even store the number "46" coded as "3032" but this would be an inefficient way to store numbers (as well as confusing). As you might have guessed, this representation of symbols by a number code is analogous to the "A" format in TUTOR variables. There are two things to note here.

(1) You must specify what "format" is being used before it is possible to know if a dial setting of 0102 represents the integer "102" (I format) or is a code for the letters "AB" (A format).

(2) Numerals in "A" format will not necessarily have the same dial setting that the identical digits in "I" format would. Even if we altered the "A" format code so that 00=0, 01=1, 02=2,..., 11=A, 12=B,..., etc. the "A" format for the numerals "11" would be 0101 rather than 0011.

There is a third type of information we might also want to store on our dial memory device. While we can store any integer from 1 to 9999 there is a problem in storing fractional values or values larger than 9999. We might get around the problem of storing fractional values by simply defining a new "format" in which a decimal point is assumed to be present between say, the second and third digits. However, there is an even better way if we are willing to give up a little precision--that is, if we are willing to specify numbers to only 3 digits rather than 4. If we let the first digit on our dial indicate the position of the decimal point and only the remaining 3 digits indicate the number, we can store numbers ranging from 0.0001 to 99,900,000. The code for the decimal point position in this case works as follows:

<u>If the first digit is</u>:	<u>Place the decimal point</u>:
0	4 places to the left of the least significant digit
1	3 " " " " " " "
2	2 " " " " " " "
3	1 " " " " " " "
4	after the least significant digit
5	1 place to the right of the least significant digit
6	2 " " " " " " "
7	3 " " " " " " "
8	4 " " " " " " "
9	5 " " " " " " "

Thus a dial setting of "0001" would be interpreted as the decimal fraction ".0001," the setting "7253" would be interpreted as "253,000," and a setting of "9999" would be interpreted as "99,900,000." You can see now why this form of coding numbers is sometimes known as "floating point." By giving up one digit to specify where the decimal point is positioned we enormously increased the range of numbers that could be specified. This last coding format is analogous to the "F" format of TUTOR variables. Again we should note that:

(1) You must specify what "format" is being used before it is possible to know if for example, a dial setting of 0102 represents the integer "102" (I format), the code for the letters "AB" (A format) or the floating point number ".0102" (F format).

(2) Numbers in one format will not necessarily have the same dial setting as the same number in another format. Thus "36" might be coded as "2932" in A format, "0036" in I format, and "4036" in F format.

While it is possible to store information in one format and interpret it in another format you can see that this is not generally a very useful procedure.

The technique PLATO uses to store information in TUTOR variables is similar to that outlined in our dial analogy. PLATO uses "dials" that allow integers with up to 14 decimal digits. Actually, numbers as large as positive or negative 140,737,488,355,327 may be stored in certain cases with TUTOR variables in the I format. In the A format you may store up to 8 characters (letters, numerals, punctuation marks, etc.) and in the F format you may store numbers with slightly better than 10 decimal digit accuracy that range from 10^{-308} to 10^{+308} (the number "1" preceded by a decimal point and 307 zeroes in the first case or followed by 308 zeroes in the second case). In practice, the possible size of stored integer or floating point numbers far exceeds the usual requirements for computations associated with instruction. Thus, restrictions have been made in certain situations to simplify presentations or computations. For example, the numbers handled by the CALC command are limited to values of 10 decimal digits or less. Another example is the SHOW command which shows a maximum of 8 decimal digits of an integer TUTOR variable or 8 digits followed by a decimal point and 7 fractional digits for a TUTOR floating point variable. Thus, 99,999,999 is the largest integer TUTOR variable which could be presented by the

SHOW command and 99,999,999.9900000 is the largest fractional or "floating point" variable which could be presented by the SHOW command (note that only the first 10 digits of the floating point variable will be accurate). As you can see, numbers of this size or larger would probably be undesirable in a teaching situation because of the possbility of the student making reading errors.

If you really need the full storage and number-handling capabilities of the computer, there are generally techniques available in TUTOR to give you this ability at a slight cost in authoring convenience. For example, the ICALC and FCALC commands do not have the 10 digit limitation of the CALC command and are performed at a much higher speed than the CALC command. The drawback to using ICALC or FCALC is that these commands do not have several of the options (such as square root, logarithm, sine or cosine functions) available to the user of the CALC command. Also, CALC permits a complex, multiple-variable equation to be written as a single equation while it would generally be necessary to break this same equation into several parts before ICALC or FCALC could be used.

To review, you have now seen the reason why it is necessary to specify the format of TUTOR variables as they are used. You may alter this format at any time--either when storing information or when retrieving it--but you have seen that it is generally best to interpret information in the same format in which it was stored. You may use TUTOR variables to store information over the entire period of the lesson (as, for example, for keeping a record of the students' first names in order to give an "individualized" response to his

ANSWERS) or you may want to use variables as a sort of scratch pad by students for keeping track of intermediate steps in lengthy problems.

Alteration of TUTOR Variables

Chapter 9 will explain more about where and how TUTOR variables for a particular student are stored. For the moment it will be enough to know that these storage spaces are available to the author even when a student is not working on a lesson. In general, there are three ways be which TUTOR variables are usually altered:

(1) By the author prior to the lesson (e.g. to preset certain lesson options for particular students).
(2) During the lesson as a result of encountering calculation type statements (e.g. a variable is increased by one when a student response matches the tag of an ANS command)
(3) During the lesson as a direct result of a student response (e.g. a student response is placed in a variable by use of a STORE statement).

Alteration of variables by the author will be explained in Chapter 9 as a part of the description of author operations. The balance of this chapter will outline the last two methods of variable alteration.

Calculation commands may be positioned in a lesson so that they will function as UNIT, ARROW and ANS Contingencies. Such commands then generally produce an alteration of TUTOR variables when activated. If the contingency which they are part of never occurs, then the alteration which they direct will also not occur. In a sense then, TUTOR variables can be made to act as flags or signals which indicate that a particular student has passed through a particular part of a lesson.

The CALC command may be considered as the prototype of calculation commands. The statement,

```
CALC    I9=3
```

alters variable "9" by placing the integer "3" (interpreted in "I" format) in it. The use of a variable as a counter leads to frequent use of statements such as,

```
        CALC     I9=I9+1
```

which alters variable "9" by replacing the number formerly stored there with the next higher integer. The frequent use of variables as counters has prompted several "short-cut" commands

"Short-cut" Statement	Equivalent CALC Statement
ADD1 I8	CALC I8=I8+1
SUB1 I12	CALC I12=I12-1
ZERO I3	CALC I3=∅

It should be clear that commands such as ADD1 are only conveniences which could be replaced by the more general CALC command. At another level, it is sometimes desirable to perform large numbers of calculations as a part of a single contingency. For example, the computer could be used to solve a fixed type of problem for the student. In such a situation it is convenient to have a command which performs operations in the most rapid manner possible. Even though a single calculation might be done very rapidly, large numbers of calculations could cause undesirable delays for the student. The ICALC and FCALC commands perform a restricted number of the CALC operations. However they perform these operations much more rapidly than CALC (there are certain other advantages to use of ICALC and FCALC as well). Again it should be clear that an understanding of the uses of the CALC command is basic to the use of most calculation commands. The individual descriptions of calculation commands in the yellow section of this manual detail limitations and advantages of each command as well as give simple examples of their use. Chapter 11 has several more advanced examples of the use of variables in lessons.

The last typical way by which variables are altered is by direct action of the student. Two commands, STORE and STORA, are used as ARROW contingencies. STORE places the student's response directly in a specified variable under a specified format. STORA allows the student to type a mathematical expression which is evaluated by PLATO. The value of this expression is stored in a specified TUTOR variable. If the contents of the variable are then displayed on the student's screen by a SHOW statement, the student is able to use PLATO as a desk calculator. The reader is again directed to the yellow section of this manual for a more complete description of the STORE and STORA commands.

PEEK, A Diagnostic Routine

Authors often find helpful, especially in testing lessons which contain involved calculation, the ability to examine and alter TUTOR variables while acting as a student. The Units shown below may be temporarily inserted at the end of any lesson for this purpose. They also serve as a practical example of several of the options which will be explained in Chapters 7 and 8.

The author may reach the diagnostic routine from any point in his lesson by pressing the TERM key, typing the word "PEEK," and pressing the NEXT key. Once in the routine the author specifies any legal format ("A", "I", or "F") and variable number (1-63). The routine shows the value of the contents of the specified variable as interpreted by the desired format. The author can then alter the value of the variable, look at another variable, or return to the lesson.

```
UNIT     PEEK
TERM     PEEK
C        DIAGNOSTIC UNIT, USES I52, I53, AND I54
WHERE    301
WRITE    WHICH VARIABLES? (INCLUDE FORMAT LETTER)
WHERE    1007
WRITE    PRESS -NEXT- WHEN FINISHED
ARROW    519
INHIB    OKNO
LONG     1
ANS      A
ICALC    I53=0-1
ANS      I
ZERO     I53
ANS      F
ICALC    I53=1
WRONG
WHERE    601
WRITE    YOU MUST INCLUDE A LEGAL FORMAT CODE
         (EITHER -A-, -I-, OR -F-)
         ERASE AND TRY AGAIN
ARROW    521
INHIB    ARROW
STORE    I52
ANS
GOTO     I52,ILL,ILL,X
ICALC    I54=63-I52
GOTO     I54,ILL,X
JUMP     PEEK2


UNIT     PEEK2
WHERE    410
JOIN     I53,FORMA,FORMI,FORMF
WHERE    401
WRITE    VARIABLE
         NOW HAS THE VALUE
WHERE    405
SHOW     I52
WHERE    801
WRITE    PRESS -A- TO CHOOSE ANOTHER VARIABLE
         -BACK- TO RETURN TO THE LESSON
         -C- TO ALTER THIS VARIABLE
ARROW    1120
LONG     1
ANS      A
JUMP     PEEK
ANS      C
ARROW    1420
JOIN     I53 STA,STI,STF
WHERE    1401
WRITE    WHAT IS NEW VALUE?
         (ENTER AND PRESS -NEXT-)
ANS
JUMP     PEEK2
```

```
UNIT    FORMA
WRITE   A
WHERE   519
SHOW    A(52)

UNIT    FORMI
WRITE   I
WHERE   519
SHOW    I(52)

UNIT    FORMF
WRITE   F
WHERE   519
SHOW    F(52)

UNIT    STA
STORE   A(52)

UNIT    STI
STORE   I(52)

UNIT    STF
STORE   F(52)

UNIT    ILL
JUDGE   NO
WHERE   601
WRITE   YOU CHOSE A NON-EXISTANT VARIABLE
        ERASE AND TRY AGAIN
```

Chapter 7

ASSIGNED OPERATIONS

Certain TUTOR commands allow a very flexible type of operation which is termed an "assigned operation." Normally each command specifies a single operation which is determined at the time the lesson is written. For example,

```
JUMP   BD7
```

indicates that the student will be immediately moved ("JUMPed") to the UNIT whose title is "BD7" when he comes to the point in the lesson where the JUMP command is encountered.

Now suppose we wanted the student to go to a different UNIT for each of three possible answers to a question. It would be possible to do this with three separate JUMP commands properly placed in the lesson. However, an assigned operation would permit the same thing to be done with a single JUMP command.

```
UNIT      TRAFFIC                    UNIT      TRAF
WRITE     A YELLOW LIGHT MEANS       WRITE     A YELLOW LIGHT MEANS
          1)  SPEED UP                         1)  SPEED UP
          2)  STOP                             2)  STOP
          3)  SLOW DOWN                        3)  SLOW DOWN
ARROW     501                        ARROW     501
ANS       1                          STORE     I5
JUMP      BD7                        ANS
ANS       2                          JUMP      I5,X,X,BD7,BD11,BD9,X
JUMP      BD11                       JUDGE     IGNORE
ANS       3
JUMP      BD9
ANS
JUDGE     IGNORE
```

UNITs TRAFFIC and TRAF do exactly the same thing but UNIT TRAF uses 4 fewer commands by using JUMP with the assigned operation option. Look closely at the tag of the JUMP command in UNIT TRAF. Notice that it consists of a TUTOR integer variable followed by a list of "titles" which are separated from each other by commas.

```
JUMP  I5,X,X,BD7,BD11,BD9,X
```

TUTOR variable I5 contains the answer of the student (this was done by the STORE command). PLATO looks at the tag of the JUMP command and sees that it must be an assigned operation, since it contains a TUTOR integer variable and a list of titles separated by commas PLATO then finds the value of I5 and JUMPs to one of the UNITs mentioned in the following list of titles. The titles "BD7", "BD11" and "BD9" are names of UNITs in the same lesson. The title "X" is a dummy title which tells PLATO to forget about the JUMP command and simply go on to the next TUTOR command. The dummy title "X" should be used only as a substitute for a UNIT title. If I5 is any negative number the JUMP is made to the first title in the list Since the first title is "X", the JUMP command is ignored if the student enters a negative answer. If I5 is zero, the JUMP is made to the second title in the list (again an "X" in this example). If I5 is 1, the JUMP is made to the third title; if I5 is 2, the fourth title; etc The sixth title in our example is an X and this title would be chosen if I5 was 4 or greater The list could have been made longer or shorter as desired Notice that a JUMP will be made in our example only if I5 contains a 1, 2 or 3. If I5 has any other value the JUMP is ignored and the next command in the UNIT is used by TUTOR. In our example the next command is JUDGE with an IGNORE tag which would cause the student's answer to be erased and ignored by PLATO. Incidentally, it should be obvious that it would be unwise to ever name one of your UNITs as "UNIT X."

The complete description of a particular TUTOR command tag will indicate if assigned operations are permitted with that command The list of titles in the tag of an assigned operation is not always a list of UNITs. In the case of the JUDGE command, for example, the list consists of judging options. Each of these options could be selected from the list if the TUTOR integer variable has a certain value. For example in

```
JUDGE  I3,NO,IGNORE,OK,NO,OK
```

if I3 is	the student's answer is
negative	judged NO
zero	erased and ignored
1	judged OK
2	judged NO
3 or greater	judged OK

The dummy Unit title "X" would not, of course, be used in the JUDGE command since the tags of a JUDGE command are not Unit titles.

If an assigned operation command contains titles of Units which are not available, e.g. where the Unit has not yet been written, PLATO will act as if the dummy Unit title "X" was present instead of the title of the unavailable Unit.

All commands which use the assigned operation option can also use the indirect referencing option explained in Chapter 8.

Chapter 8

INDIRECT REFERENCING BY TUTOR VARIABLES

TUTOR commands which use a variable in their tag have a very useful option known as "indirect referencing." An example will clarify its use.

EXAMPLE: In the command

```
CALC  I20=I1 + I2
```

suppose that I1=4 and I2=5. Then I1 and I2 could be said to "directly refer" to the numbers 4 and 5 respectively. This would be an instance of "direct referencing" by TUTOR variables.

Now, in the command

```
CALC  I20=I(1) + I(2)
```

the parentheses around the 1 and the 2 indicate that I1 and I2 refer <u>indirectly</u> to specific values. The numbers within the parentheses refer to TUTOR integer variables which have as their value a number from 1 to 63. Since in this example I1=4 and I2=5, this second CALC command tells PLATO to put the sum of the values of TUTOR variables I4 and I5 in variable I20. That is, variables I1 and I2 "indirectly refer" to the numbers contained in variables I4 and I5. This is an instance of "indirect referencing" by TUTOR variables.

In other words, if

I1=4, I2=5, I4=22, and I5=23,

then `CALC  I20=I(1) + I(2)`

is interpreted by PLATO as

```
CALC  I20=I4 + I5
```

or `CALC  I20=22 + 23`

An analogy may be helpful in understanding indirect referencing. Direct referencing is similar to looking up the word "feline" in a dictionary and finding the word's definition under the entry "feline." Indirect referencing would be similar to looking up the word "feline" under the entry "feline" and finding a note telling you that the definition will be found instead under the entry for "cat." In indirect TUTOR referencing the "note" (telling you where the desired value is located) is the value of a TUTOR integer variable.

The analogy breaks down when we come to the formats of the variables concerned. However, it should be apparent that for most cases we will want the formats of the referencing and referenced variables to match. Thus,

for A(20), if I20=5, variable 5 should be defined as A5;

for F(32), if I32=7, variable 7 should be defined as F7; and

for I(22), if I22=9, variable 9 should be defined as I9.

In other words, the variable whose number is enclosed within the indirect referencing parentheses must always be defined as an integer variable (I20, I32, I22, in the example above). The variable referenced by this integer variable should in turn be defined as a variable with the same format as the original referencing variable (A5 for A(20), F7 for F(32), and I9 for I(22) in the above example).

Indirect referencing is very useful when a formula or UNIT using TUTOR variables is used many times in the same lesson. Indirect referencing allows a single UNIT to be used in all applications without forcing the author to redefine the variables used each time.

As an example of the use of indirect referencing, suppose that an author wishes that the value of TUTOR variables I11 through I25 be set to zero before he uses them during a portion of a lesson. Perhaps the author will be using these 15 variables to keep a count of 15 different types of errors a student might make. Obviously, erroneous results might occur unless any old information stored in these variables was first "erased." Both units ZERO and ZRO produce the desired zeroing of I11 through I25. In Unit ZRO, assume that TUTOR variable I9 is preset to the value 11.

```
UNIT      ZERO                UNIT      ZRO
ZERO      I11                 ZERO      I(9)
ZERO      I12                 ADDI      I9
ZERO      I13                 ICALC     I10=I9-26
ZERO      I14                 GOTO      I10, ZRO, X
ZERO      I15                   .
ZERO      I16                   .
ZERO      I17                   .
ZERO      I18
ZERO      I19
ZERO      I20
ZERO      I21
ZERO      I22
ZERO      I23
ZERO      I24
ZERO      I25
  .
  .
  .
```

The effect of Unit ZERO should be obvious but that of Unit ZRO may not be so readily apparent. Let us go through Unit ZRO step by step to see how PLATO would act when it encountered these commands. In a previous UNIT, the author has placed the number 11 in variable I9. Upon entering Unit ZRO, PLATO thus interprets ZERO I(9) as "ZERO I11." As a result, any prior information that might have been stored in TUTOR variable I11 is replaced by the number zero. The ADD1 command then increases the value of I9 by one, giving I9 the value 12. Since we are really only interested in the point at which I9 exceeds 25, the temporary variable I10 is used. I10 is defined by the operation I10=I9-26. Thus, I10 is negative unless I9 exceeds 25.

Now, when PLATO goes through Unit ZRO a second time ZERO I(9) is interpreted as "ZERO I12", the value of I9 is increased to 13 and a new GOTO to the beginning of Unit ZRO is made. This "looping" through Unit ZRO continues until, on the 15th time, TUTOR variable I25 is set to zero, I9 is increased to 26, I10 becomes zero and the GOTO command is not performed. Commands following the GOTO statement are now executed.

Chapter 9

OPERATING AS AN AUTHOR

A PLATO student station is the main means of communicating with PLATO. The student, of course, uses a student station to receive lessons. When a station is being used for lesson presentation, it is said to be operating in STUDENT MODE. Any PLATO student station can also be used by a teacher to produce or alter lessons or to give special instructions to guide lesson presentation. When a station is used in this latter manner it is said to be operating in AUTHOR MODE. A station can be quickly shifted from either of these modes of operation to the other. Each station operates independently and any mixture of "authors" or "students" may simultaneously use PLATO with each acting almost as if he were the only person using the system.

The user of a student station in AUTHOR MODE has a great deal of power. He can alter virtually any lesson available as well as control student access to lessons. It is therefore necessary to limit use of AUTHOR MODE to responsible individuals. This point cannot be emphasized too strongly. Never let any student or other unauthorized individual see how you shift a station to AUTHOR MODE. Likewise, you should never leave a station unattended while it is in AUTHOR MODE. It should go without saying that careless actions of an authorized person can also cause a great deal of damage.

Entry to AUTHOR MODE

The general method of entry to AUTHOR MODE at any keyset is by pressing the TERM key and then typing a "code word." This code word will be changed from time to time to insure the security of lessons and student information. You will be told the current code word if and when you have a valid need to use AUTHOR MODE. If you must write this word down to remember it, please do so in a place or a fashion that will not identify its use to an unauthorized person.

To enter AUTHOR MODE from an unused student station

1. Press TERM key
2. Type code word (main AUTHOR MODE display will appear)

To enter AUTHOR MODE from a station in use by a student

1. Press TERM key ("WHAT TERM?" will appear on bottom line of screen)
2. Type "FINISHED" and press NEXT key (WELCOME TO PLATO message will appear)
3. Type code word (main AUTHOR MODE display will appear)

The first two steps in the procedure for entry from a station in use by a student cause the student's current location in the lesson to be stored away. When you have finished using AUTHOR MODE and return the station to STUDENT MODE, he can "sign in" again and he will be returned immediately to his last location in the lesson. He could also "sign in" at any other station or at some later time and immediately take up where he left off. Thus, the TERM-"FINISHED"-NEXT procedure is generally useful in cases where the student must leave before he has completed an entire lesson. You should, of course, not allow the student to observe you as the code word is typed.

Exit from AUTHOR MODE

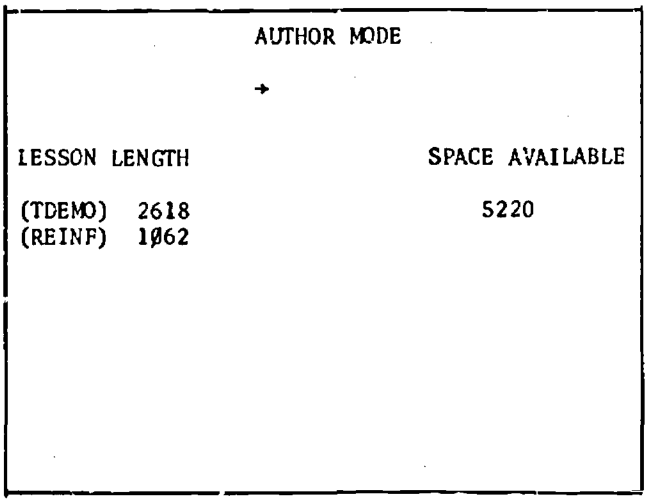
The main AUTHOR MODE display serves as both the exit and entrance point to AUTHOR MODE. An author will always be able to return to the main AUTHOR MODE display from the various AUTHOR MODE options by pressing key BACK one or more times. Once on the main AUTHOR MODE display, pressing key BACK once will shift the station into STUDENT MODE. Authors should always return a station to STUDENT MODE before leaving.

To enter STUDENT MODE from AUTHOR MODE

1. Get to main AUTHOR MODE display by pressing key BACK one or more times.
2. Press key BACK (station shifts to STUDENT MODE. WELCOME TO PLATO message appears if student operation is permitted. SESSION FINISHED message appears if student operation is not permitted).

Main AUTHOR MODE Display

Figure 9.1 shows the main AUTHOR MODE display. This display is the access point to all major AUTHOR MODE options. Options are selected by typing the code word of a desired option and pressing the NEXT key to activate that option. As the code word is typed, it appears on the screen after the small arrow. The available options will be discussed in the next section of this chapter.

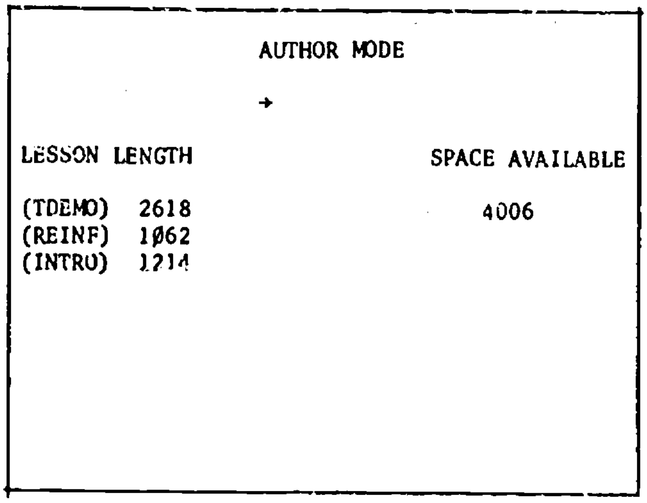


```
                    AUTHOR MODE

                    →

LESSON LENGTH                        SPACE AVAILABLE

(TDEMO)   2618                             5220
(REINF)   1Ø62
```

Figure 9.1

Main AUTHOR MODE display. Two lessons, (TDEMO) and (REINF), are indicated as being available for student use. In addition, 5220 "words" of storage space are available for use by other lessons.

The AUTHOR MODE display shows what lessons are currently available to students during a particular class period and how much space is available for additional lessons. In order for a lesson to be available for use by students, a copy of the lesson must be placed in (or "read into") a section of the computer. The total number of lessons that are available to students at any one time is limited by the amount of space available in the computer. Computer space is measured in "word" units. Figure 9.1 shows an example in which two lessons have been placed in the computer. Lesson (TDEMO) takes up 2618 words and lesson (REINF) takes up 1062 words. There are 5220 words still available to other instructors. The lesson author need never worry about how lesson length is measured since the main AUTHOR MODE display tells him all he will ever need to know. The first time a new lesson is placed in the computer, the author should make a note of its length (as indicated on the AUTHOR MODE display). Thereafter, the author need only verify that the space available in the computer equals or exceeds the length of desired additional lessons. For example, during a class period in which lessons (TDEMO) and (REINF) were already available to students, an author might decide that lesson (INTRO) should also be made available. Having used lesson (INTRO) before, the author would know that (INTRO) takes up 1214 words of space. Since 5220 words are available (see Figure 9.1), there is ample room for lesson (INTRO). After "reading in" lesson (INTRO), the main AUTHOR MODE display would appear as in Figure 9.2. The space available for additional lessons has been reduced to 4006 words.



```
                      AUTHOR MODE

                      →

LESSON LENGTH                       SPACE AVAILABLE

(TDEMO)   2618                          4006
(REINF)   1Ø62
(INTRO)   1214
```

Figure 9.2

Main AUTHOR MODE display after addition of lesson (INTRO).

Lessons may be added to or removed from the computer at any time by use of various AUTHOR MODE options which will be described shortly.

One caution should be observed in using the information on the AUTHOR MODE display. The information is current only at the time it is first displayed upon entry from STUDENT MODE or return from an AUTHOR MODE option. Thus, your AUTHOR MODE display will not reflect additions or deletions made by another author a few seconds after your current copy of the message was first displayed.

AUTHOR MODE Options

As indicated previously, an AUTHOR MODE option is selected by typing its code word while on the main AUTHOR MODE display. It is activated by pressing key NEXT. If an error is made during typing, the ERASE key may be used at any time before the option is activated. If an author attempts to activate a misspelled or nonexistant option, his station will be immediately shifted to STUDENT MODE (it is assumed that STUDENT MODE is the best place to put people who make mistakes while in AUTHOR MODE). However, the author should not rely on this feature to catch his careless mistakes. Several special AUTHOR MODE options (not described here) are reserved for use by TUTOR and PLATO system personnel. Activation of some of these special options at certain times could be catastrophic. Therefore, the author should use particular care while selecting AUTHOR MODE options and should certainly never attempt to use options which are not described here. Eight AUTHOR MODE options are available to the general TUTOR author: EDIT, READIN, DATA, START, STOP, DELETE, PRINT, and RECORDS. Each of these will be discussed in detail in the following sections.

(1) The EDIT Option: The EDIT option permits the author to write a lesson in a form which is immediately usable. As you have seen in prior chapters, a TUTOR lesson consists of a collection of units, which in turn are made up of a sequence of TUTOR statements. In the EDIT option, the author can enter or alter the list of TUTOR statements which make up his lesson. The author does this by typing each TUTOR command and its tag on the keyboard at a PLATO station. As the author types, the commands are sent to the computer which later places them on a magnetic disk storage device. Each lesson is assigned a specific area on one of these disk storage devices.

Lessons are identified by abbreviated titles which consist of up to six letters or numbers enclosed in parentheses. (TDEMO) and (REINF) are two such titles. Let us observe how an author might start a new lesson entitled (BOSH).

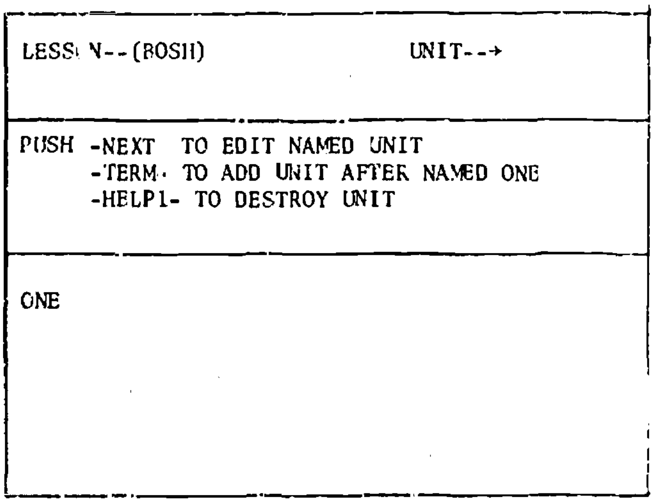
The first steps in beginning a new lesson consist of (a) writing a draft of the lesson and (b) getting permission from PLATO personnel to place the lesson on a magnetic disk storage pack. If permission is granted, an area of a pack will be set aside for the author under the title which he requests. After the area has been set aside the author is notified. He may then begin writing his lesson using the EDIT option of AUTHOR MODE.

To EDIT a lesson

1. Get to the main AUTHOR MODE display
2. Type EDIT and press key NEXT (the AUTHOR MODE display will disappear and a new message will request the name of the lesson to be EDITed)
3. Type title of lesson to be EDITed and press key NEXT (EDIT index display will appear)

NOTE: Remember that lesson titles are always enclosed in parentheses.

Figure 9.3 shows the EDIT index display for lesson (BOSH) before the author has written anything in the area reserved for (BOSH). The top line indicates that EDITing is being done in the area reserved for lesson (BOSH).

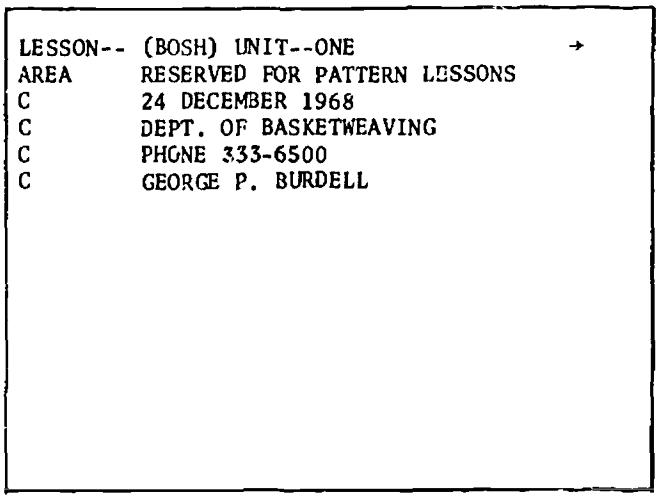


```
LESSON--(BOSH)                    UNIT--→

PUSH -NEXT  TO EDIT NAMED UNIT
     -TERM- TO ADD UNIT AFTER NAMED ONE
     -HELP1- TO DESTROY UNIT

ONE
```

Figure 9.3

EDIT index display for lesson (BOSH) before the author has written any Units.

The next three lines give directions for either (1) editing a Unit already present, (2) adding a new Unit, or (3) destroying a Unit. Since (BOSH) is a brand new lesson, the only Unit present is one placed there temporarily by PLATO personnel when they reserved the lesson space for him. This temporary Unit is titled "ONE". If the author wishes to inspect Unit One he types "ONE" on his keyboard. The word "ONE" will appear on the first line of the EDIT index display after the word "UNIT". Pressing key NEXT allows the author to edit this Unit. Figure 9.4 shows the contents of Unit ONE which the author will see on his screen. The Unit presently consists of five statements, an AREA statement indicating the general topic of the lesson and four "C" statements giving information about the author and the date the space was reserved. The "C" command is a "dummy" command which is ignored during a TUTOR lesson but can be used in AUTHOR MODE to store useful information for the author or PLATO personnel.

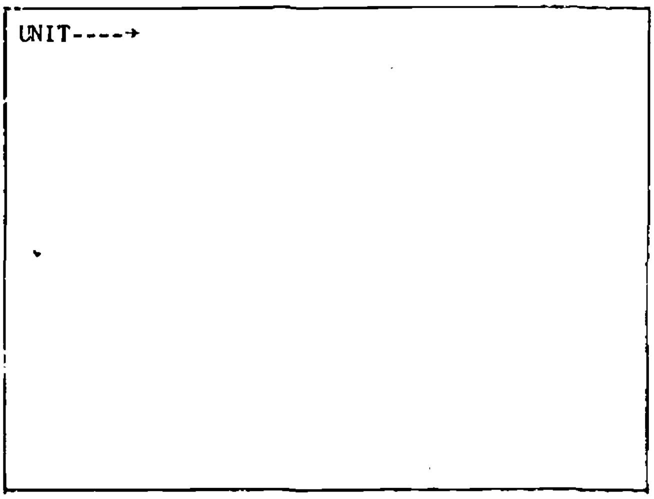


```
LESSON--  (BOSH) UNIT--ONE                     →
AREA      RESERVED FOR PATTERN LESSONS
C         24 DECEMBER 1968
C         DEPT. OF BASKETWEAVING
C         PHONE 333-6500
C         GEORGE P. BURDELL
```

Figure 9.4

Unit Contents - as seen by author after typing "ONE" and pressing key NEXT. The author returns to EDIT index display by pressing key BACK.

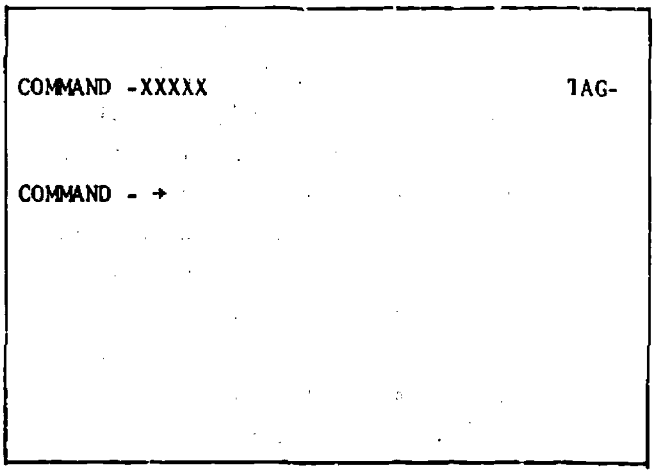
In this situation, the "C" statements are used to [illegible] personnel about the lesson and the author. Now, suppose the author wished to alter this information and also wished to call his first Unit "B-1" rather than "ONE". By pressing key BACK he could return to the EDIT index display. He could then add a Unit after Unit ONE by typing "ONE" and pressing key TERM as per the instructions on the EDIT index display. (Note--if TERM is pressed when no Unit name has been entered, the new Unit will be inserted in front of all other Units at the beginning of the lesson.) A new message would then request the name of this new Unit (see Fig. 9.5). After typing "B-1", the author presses key NEXT to indicate that he has finished typing the name of the new Unit.



```
UNIT----→
```

Figure 9.5
Request for name of new Unit

The display shown in Figure 9.6 would then appear. The X's in the line displayed at the top indicate that material is being inserted at the start of a Unit. At this point the author begins producing the contents of his lesson.



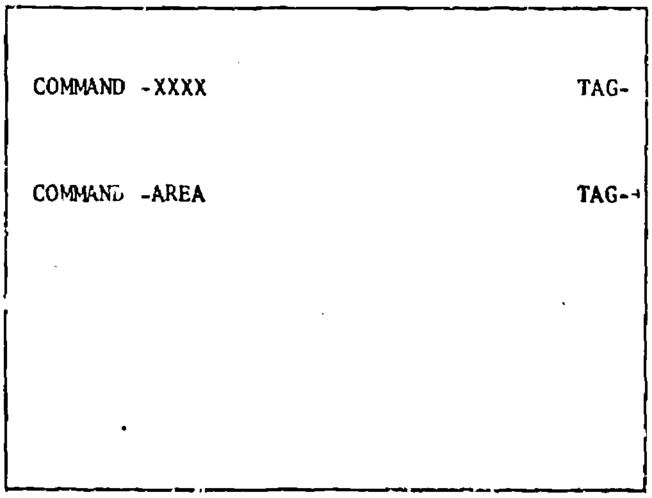
```
COMMAND -XXXXX                              TAG-

COMMAND - →
```

Figure 9.6

If the first statement in Unit B-1 was:

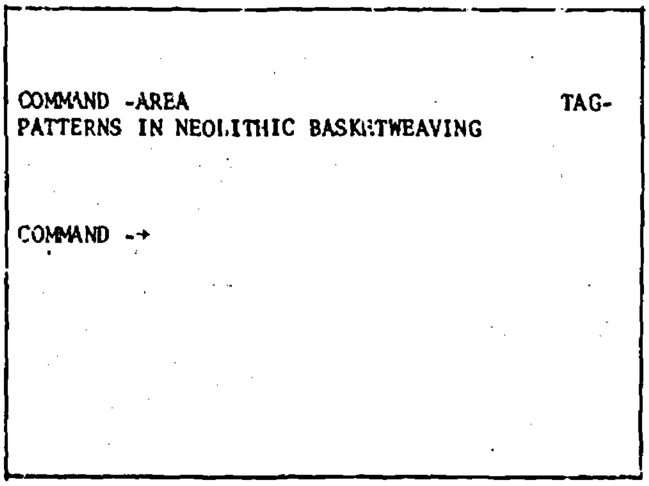
AREA PATTERNS IN NEOLITHIC BASKETWEAVING

the author would type "AREA" and press key NEXT to indicate that he was finished with the command portion of the statement. The display would shift to that shown in Figure 9.7. After typing the tag of the AREA statement and pressing NEXT the screen would shift to Figure 9.8.



```
COMMAND -XXXX                                   TAG-


COMMAND -AREA                                   TAG-→
```

Figure 9.7



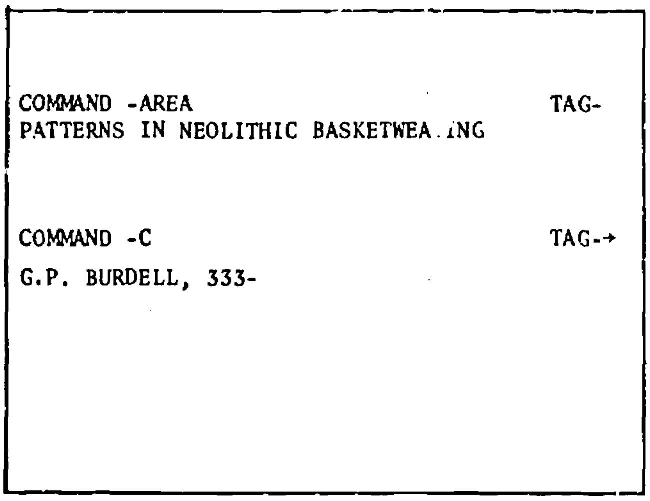
```
COMMAND -AREA                                   TAG-
PATTERNS IN NEOLITHIC BASKETWEAVING


COMMAND -→
```

Figure 9.8

Suppose the next statement was:

C G.P. BURDELL, 333-6500

After the command "C" had been entered, and the author had started writing the tag, the screen would appear as in Figure 9.9. Note that the immediately preceding statement is retained on the screen. Mistakes in typing may be corrected by use of the ERASE key if detected before key NEXT is pressed.

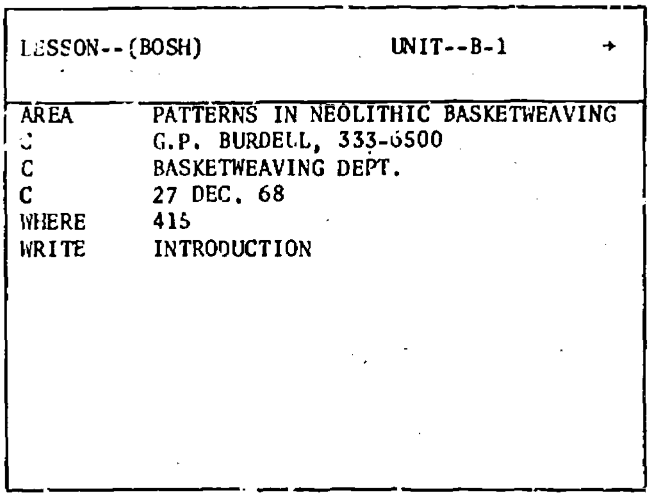


```
COMMAND -AREA                                TAG-
PATTERNS IN NEOLITHIC BASKETWEA.ING


COMMAND -C                                   TAG-→
G.P. BURDELL, 333-
```

Figure 9.9

Once NEXT has been pressed, however, the entire command containing the error must be deleted. Such deletions are performed by use of one of several sub-options to be explained next.

After entering several statements the author might wish to check over all of the statements he has entered in the Unit. By pressing key BACK he will be able to see the contents of his Unit as shown in Figure 9.10.



```
LESSON--(BOSH)                  UNIT--B-1        →

AREA       PATTERNS IN NEOLITHIC BASKETWEAVING
C          G.P. BURDELL, 333-6500
C          BASKETWEAVING DEPT.
C          27 DEC. 68
WHERE      415
WRITE      INTRODUCTION
```

Figure 9.10

Unit Contents for Unit B-1

While on a Unit content display the author has six sub-options available. He may:

1. Return to the lesson index display (by pressing key BACK),
2. Move the display forward (up) by a specified number of statements,
3. Move the display backward (down) by a specified number of statments,
4. Delete a specified number of statements,
5. Insert additional statements, or
6. Save a specified number of statments.

Note that a small arrow appears in the upper right corner of the Unit content display shown in Figure 9.10. If the author presses letter or number keys while the Unit content display is present, the corresponding characters will appear on the screen following the small arrow. Sub-options 2 through 6 are activated as follows:

Sub-options available on Unit Content Display

1. To move the display FORWARD (up), type "F" and a number. Activate by pressing NEXT (display moves up by the specified number of lines).

2. To move the display BACKWARD (down), type "B" and a number. Activate by pressing NEXT (display moves down by the number of lines specified.

3. To DELETE lines, type "D" and a number. Activate by pressing NEXT (beginning with the top statement, the specified number of statements will be deleted).

4. To INSERT new statements, type "I". Activate by pressing NEXT (insertion will begin following the top statement on the display).

5. To SAVE a number of statement lines, type "S" and a number. Activate by pressing NEXT (beginning with the top statement, the specified number of statements will be saved).

Examples: F3, B2, D6, I, S2. NOTE--Any letter activated without a following number is interpreted as if the number "1" were present. Thus F and F1 will both move the display up one line.

Following use of the Delete sub-option, any preceding statement which was formerly not on the display will drop into view (and the deleted statement will of course disappear). The Insert sub-option also may be used with a number to indicate that the insertion is to occur following a specified statement counting from the top statement on the display. However, it is generally safer for inexperienced authors to shift the display so that insertions are always made after the top statement. The Insert sub-option moves the author to a type of display like that seen in Figure 9.8. Actually upon beginning a new Unit the author is automatically placed in the Insert sub-option. Writing a new Unit or adding to the end of an unfinished Unit are simply special cases of "inserting" new statements.

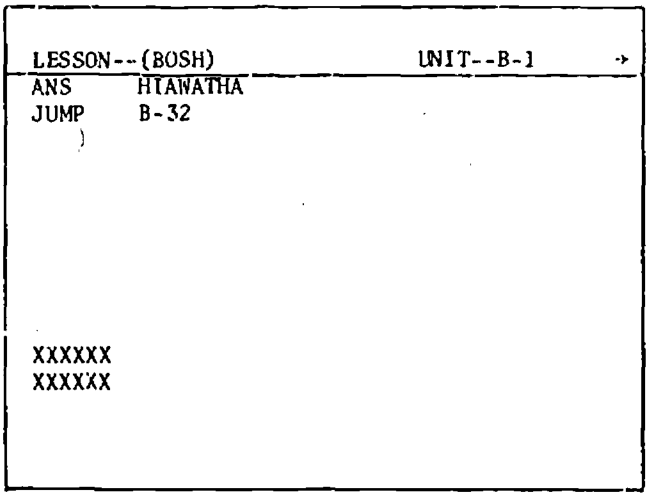
The total number of lines that can be saved with the Save sub-option varies depending upon how much information the lines contain. "Saved" information can be inserted anywhere in a lesson by suffixing an I directive with the letter S. For example, if the author types in the directive IS, the previously "saved" lines are inserted after the first line of the unit content display. Information saved via an S directive remains available until destroyed by another S directive or until the author leaves the EDIT option. Since the storage space for saving information is limited, authors attempting to save more than a few lines should always be careful to check that all the information they requested has actually been saved.

To correct errors in Units which have already been written, the Delete and Insert sub-options are used in conjunction. To avoid compounding errors, authors are urged to adopt a systematic approach to correcting lesson errors. One such approach is outlined here. Many authors discover (the hard way) that little is gained and much can be lost when an erroneous statement, or possibly what the author thought was an erroneous statement, is deleted before, rather than after, its replacement has been inserted.

To correct erroneous statements

1. Get to Unit Content Display
2. Move incorrect statement to top of display
3. Insert correction
4. Return to Unit Content Display by pressing BACK (incorrect statement will be on top line, correction will appear below).
5. Verify that correction is actually correct
6. Delete incorrect statement (type "D", press NEXT)

Only about 17 statements will be visibl. a time on a Unit content display. Thus, on lengthy Units it will not be possible to see the entire Unit without using the "F" sub-option. There is a maximum permissible size for single Units; however, this limit will rarely be encountered by most authors. An author who is approaching this limit will see a row of X's appear on the Unit content display as shown in Figure 9.11. Figure 9.11 of course shows only the last few statements of a very long Unit.

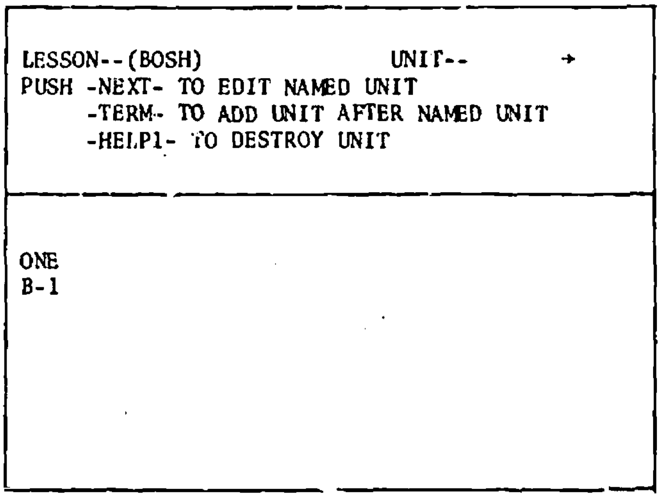


```
LESSON--(BOSH)                      UNIT--B-1       ->
ANS      HIAWATHA
JUMP     B-32
   )






XXXXXX
XXXXXX
```

Figure 9.11
Unit content display for Unit which is nearly "full".

The limitation in maximum length of a Unit need not be of practical concern to an author since portions of the draft version of the Unit may themselves be defined as Units and JOINed into the overly lengthy Unit at the appropriate point (see Chapter 5).

To return to our example, suppose that the author has satisfactorily completed his Unit B-1 through appropriate use of the "F", "B", "D" and "I" sub-options of the EDIT option. After reviewing his typing by inspection of the Unit content display, the author returns to the lesson index display by pressing key BACK. In general, throughout AUTHOR MODE, an author may always "back out of" the various levels of the options by use of the BACK key. Upon return to the lesson index display, the author observes that the display now appears as Figure 9.12. Since Unit B-1 replaces Unit ONE, the author will now wish to eliminate Unit ONE.



```
LESSON--(BOSH)                UNIT--       →
PUSH -NEXT- TO EDIT NAMED UNIT
     -TERM- TO ADD UNIT AFTER NAMED UNIT
     -HELP1- TO DESTROY UNIT

ONE
B-1
```

Figure 9.12

Lesson index display after addition of Unit B-1

This is done by following the third instruction on the display. "ONE" is typed by the author and the SHIFT and HELP keys are pressed simultaneously ("HELP1" being interpreted as SHIFT+ HELP). The lesson index display is immediately altered and the "ONE" listing disappears. The procedure and the instructions for destroying a Unit are purposely complex. The intention is to avoid accidental destruction of lesson material by an author or "malicious mischief" by a student who might get into AUTHOR MODE accidentally or through the carelessness of an author. For the same reason you will note that no instructions are given in AUTHOR MODE itself for use of the "I", "D", "S", "F" and "B" sub-options.

As the author adds additional Units, the names of these Units will appear on the lesson index display. Depending on the length of the individual Units, between 32 and 64 Units may be listed on the index display before the lesson space is filled to capacity. Experience has shown that lessons of this size are generally more than long enough to take up a typical class "hour." When Units are very short it is possible to have more than 64 Units in a lesson. If this situation arises, the author should consult PLATO personnel for special instructions.

Only one author can EDIT a lesson at a time. If an author attempts to EDIT a lesson already being EDITed by another author, he will receive a message informing him of the conflict. Pressing key NEXT will then return him to the main AUTHOR MODE display.

This completes the discussion of the EDIT option. However, before continuing on to the remaining seven options, we will try to clear up a point which may be puzzling some readers. We have referred to the fact that an author using EDIT stores his lessons on the magnetic disk. However, the distinction between disk storage and computer storage has not yet been made clear.

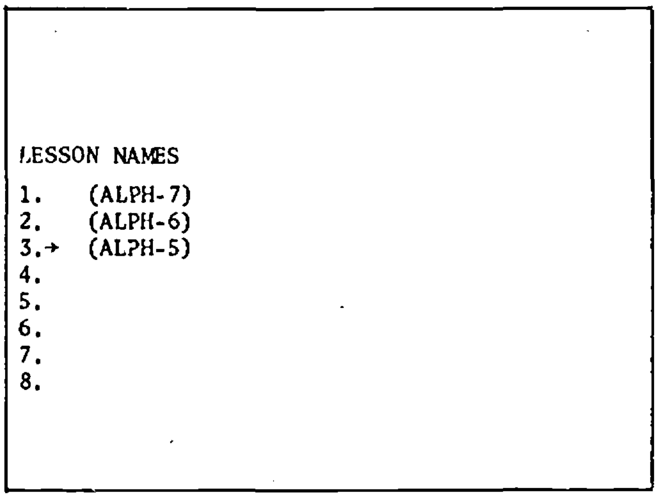
Associated with the computer is a high-speed "memory" or storage device. Information such as a TUTOR lesson stored in this high-speed memory is rapidly accessable. In fact, the speed with which particular items of information can be retrieved from this high-speed memory is measured in terms of millionths of a second. Such speeds are a necessity when large numbers of students must be served without noticeable delays. If lessons could be permanently stored in this high-speed memory, all lessons could be immediately available for student use. Unfortunately such storage devices are relatively expensive. Thus lessons are stored on a somewhat lower-speed (and less expensive)memory device except when they are actually being used by students. The magnetic disk pack is one such device. A disk pack would be far too slow on the present PLATO system for practical use during an actual lesson. However, a copy of a lesson can be "read" from a disk pack into the computer's high-speed memory in a matter of seconds. Thus the limitation in the amount of available high-speed memory space does not significantly reduce the range of lessons available to students. In other words, a group of 20 students might have quick access to any of several hundred lessons stored on disk packs but only 6-10 of these lessons would be available simultaneously because of limited high-speed storage space.

Authors may edit lessons at the same time the lessons are being used by a student. Note that what the student is using is only a copy of the lesson which was read into the computer from disk storage at the beginning of the class period. The author performs his edits on the original lesson which is always stored on the disk. Thus, changes made by the author during a class will not be apparent to the student until a new copy of the revised lesson is read from the disk storage into the computer.

2. The READIN Option. This AUTHOR MODE option allows the author to read a copy of a lesson stored on a disk pack into the high-speed memory of the computer, thus making the lesson available for student use.

> To make a lesson available for student use:
>
> 1. Get to the main AUTHOR MODE display
> 2. Verify that space is available for the lesson(s) desired.
> 3. Type "READIN" and press key NEXT (READIN message will appear - see Figure 9.13).
> 4. Type the name of each lesson desired. Use the ARROW key to move to a new line for each ad - itional lesson.
> 5. Press key NEXT to begin READIN (main AUTHOR MODE display will reappear after READIN is complete -- READIN may take up to 15 seconds if several lessons are added).

Several complications may occur during READIN. The simplest of these is that someone else might have already begun a READIN or certain other options. In this case, a message to that effect will appear. Pressing key NEXT allows a return to the main AUTHOR MODE display. A second attempt to READIN may then be made.



```
LESSON NAMES
1.    (ALPH-7)
2.    (ALPH-6)
3.→   (ALPH-5)
4.
5.
6.
7.
8.
```

Figure 9.13

READIN message. Key ARROW shifts to new line.
Press NEXT to begin READIN.

If a lesson is newly written it might well contain typing or logical errors. Certain of these errors will be detected when the lesson is read-in for student use. Most errors consist of "illegal" commands (generally misspellings of the intended commands) and lesson connections which could not be made. An example of a missing lesson connection would be a lesson containing the statement

```
NEXT      SUB3
```

when there was no Unit named "SUB3" among the lessons read in. Errors are indicated in a message which appears, when needed, immediately after READIN. The message will list the errors which were detected and the Units in which each error was found. If the author miscalculates and tries to READIN a lesson which is too large for the available space in the computer, the error message will indicate how much of the lesson did get into the computer. The author should make a note of all errors listed and then press key NEXT to return to the main AUTHOR MODE message. If unexpected error messages occur on READIN of a lesson, remove the lesson from the computer (see the DELETE option), make appropriate EDITs, and then READIN the revised lesson. Similarly, if a lesson will only partially fit into the space available, it should be removed from the computer. No harm will result from testing lessons with known or unknown errors (the lesson will operate normally up to a faulty Unit) but good authoring practice suggests that grief is minimized when known errors are corrected as they are encountered.

When several lessons are placed in the computer by the same READIN, a special feature becomes available. Connections between all lessons entered at the same time are permitted. For example, the last Unit of one lesson might contain a NEXT command allowing a connection to the first Unit of a related lesson. As long as both lessons were placed in the computer by the same READIN, this connection would be made. If the lessons were placed in the computer by two separate READINs, the connection would not be made. Under certain circumstances an author might want to READIN a single lesson even though it contained connections to other lessons. Upon READIN, an error message would inform the author of the connections which could not be made. Students could still use the lesson but the lesson would behave as if the statement (or portion of the statement) calling for the nonexistant connection was not present. If the lesson has been written so that it can function properly even without such statements, al is well. This is generally the only situation in which an author would operate a lesson for students after getting an error message on READIN.

Another advantage of using connected lessons is that duplicate Units used by several lessons need be produced only once. Examples of such duplicate Units include help, review, and other supplementary material as well as Units (known as "drivers") which allow economical presentation of large numbers of screen displays which differ only slightly from one another. As an ideal, authors should limit the total size of an individual lesson or group of lessons which must be read-in together to about 3000 words. There are several advantages to keeping a lesson or group of dependent lessons as short as possible. Among these advantages is the greater flexibility that short programs give to the instructor of a class containing students with a wide range of interests or ability levels. Equally important is the fact that during early testing of lessons it is far easier to find small amounts of "free" computer space than it is to find, say, 5000 words of unused space.

3. The DATA Option. One of the basic advantages of computer-based education is the ability to collect data as each student works his way through a lesson. By means of the DATA option, a standard data record can be saved on magnetic tape each time a student's answer is judged, each time the student presses certain control keys, or whenever an INFO command is encountered. Following the class session, these records can be sorted and printed. The DATA option applies to all users of the PLATO system. That is, if any instructor begins data collection, data is stored for all students working on the system.

To collect student data

1. Check with computer operator to verify that magnetic tape is ready for data collection
2. Get to main AUTHOR MODE display
3. Type "DATA" and press key NEXT (when "DATA" disappears from screen, data collection has begun)

Conversely, no data will be collected for any student unless some instructor asks for it. Figure 9.14 shows part of a printed copy of data from one class session.

Name	Time	Unit	Arrow	Event	Supplementary information
TENCZAR	5*42	ADD	1	WRONG	25
AVNER	4729	KEYS	2	DUP	ALFALFA
BOHN	23*51	Q23	1	INFO	256 74
TENCZAR	5*53	ADD	1		35
BLOMME	3*14	HP	1	TERM	PLATO
CRANDEY	44*42	STAT-24	1	NC	THE MEAN AND KURTOSIS.
BITZER	12*36	PROB4	1	ANS	1492
MAST	5*15	INTR	3	BACK	
STEINBER	7*15	HI	1	HELP	

Figure 9.14

Portion of a data record from a class session

Each line of data is produced at the time the event occurs. Thus, records of different students taking different lessons are all intermixed. An author would of course rarely use student data in this "raw" form. The first item in a standard data record is the student name. The second item is the time at which the event occurred. Time is measured in either minutes and seconds (e.g., 5*42 = five minutes and 42 seconds after sign-in) or in sixtieths of a second (e.g., 4729 sixtieths after sign-in) at the option of the author of each lesson. By starting each student's clock at zero when he signs-in for a class, a record of cumulative class-time can automatically be kept. The third item in a data record is the name of the Unit in which the student was working. The fourth item further specifies the student's position in the lesson by indicating which "ARROW" the student was responding to. The fifth item specifies the type of event which caused the data line to be produced. Table 9.1 lists the 16 different types of events which can produce a data record. The sixth and final item in a line of student data is supplementary information which usually consists of a record of what (if anything) the student typed.

EVENT LABEL	EVENT
none (blank)	A student response was judged "OK"
WRONG	A student response matching the tag of a WRONG or CANT statement was judged "NO"
NO	A student response which did not match any ANS-type statement tag was judged "NO"
NC	A sentence was judged incomplete
SP	A student response was judged to be misspelled
DUP	The student attempted to use the same response twice on a list
ANS	The student requested the answer
TERM	Student requested a term
INFO	An INFO command was encountered in the lesson
HELP	Student branched by pressing the HELP key
HELP1	" " " " " HELP1 "
DATA	" " " " " DATA "
DATA1	" " " " " DATA1 "
LAB	" " " " " LAB "
LAB1	" " " " " LAB1 "
HELPNO	A help-type key was pressed, but no branch was available

Table 9.1

Events which can produce a data record and the labels indicating these events

The only exceptions to this generalization occur for the ANS and INFO events. Supplementary information for the ANS event is the correct answer (if any) given the student upon his request. The supplementary information for INFO events consists of a line of information specified by the author in an INFO statement contained in a lesson.

Student data is generally sorted in the evening following classes. Authors can specify that selected data are to be kept (e.g. data for specific students or particular types of events) and how the data are to be arranged (by students, by Units, etc.) The major use of data as extensive as that shown is in revision of early forms of lessons. Two ways of using the data which give good results are (1) checking Units on which students seem to spend a great deal of time (expansion or rewriting may be needed) and (2) checking all "NO" events (responses which were not anticipated) to insure that correct responses are not being rejected and that "popular" incorrect responses receive special attention if necessary. Complete data collection is generally not warranted for final forms of lessons. If records of student performance are desired, summary data can be provided more efficiently by use of TUTOR variables and the RECORDS option discussed later.

4. The START Option. For purposes of class control, it is sometimes helpful to prevent student access to lessons which have been placed in the computer. For this reason, students will not be able to "sign-in" for any lesson until an author activates the START option. Once any author activates the START option, all students will be able to sign-in. At this point we can list the complete procedure for starting a TUTOR class session.

To start a TUTOR class session:

1. Get to the main AUTHOR MODE display
2. READIN all desired lessons (use separate READINs for independent lessons).
3. If data are to be collected, verify that a magnetic tape is ready and activate DATA
4. Type "START" and press key NEXT. WELCOME TO PLATO messages will appear on all unused stations in STUDENT MODE. Stations in AUTHOR MODE are unaffected.
5. Return your station to STUDENT MODE before leaving it (press key BACK)

If student data are not desired, step 3 should be eliminated. Other instructors can READIN additional lessons after one set of students has begun without affecting the students already at work. However, since the START option has already been activated, students belonging to the latter instructors will be able to sign-in once their lessons are read-in. If the START option is activated more than once during a class session the only effect will be to place a fresh WELCOME TO PLATO message on the screens of unused stations. Thus the procedure for starting class sessions which is outlined above may be used by instructors even if they READIN lessons after another class has begun.

The STOP Option. The procedure for ending a class session is quite simple, merely write STOP on the AUTHOR MODE display page and press NEXT. When the STOP option is activated data collection is halted and an "end" marker automatically is placed on the magnetic tape. Records of all students are also updated so that the next time they sign in they will begin where they last left off. Remember that all students are halted as soon as any instructor activates STOP. Thus, when several classes share the same time some coordination is advisable. The "press key TERM, type 'FINISHED', press key NEXT" procedure described at the first of this chapter is useful for updating records of individual students who leave before the end of the class session. This procedure also releases the student station for use by another student or by an author.

To stop a TUTOR class session:

1. Get to the main AUTHOR MODE display
2. Type "STOP" and press key NEXT. The SESSION FINISHED message will appear on all stations in STUDENT MODE. Stations in AUTHOR MODE are unaffected.

6. The DELETE Option. Lessons should be removed from the computer as soon as a class session is completed, whenever all students have finished using a particular lesson during a class session, or whenever testing of the lesson is completed. The procedure for removing a lesson from the computer's high-speed storage is outlined below.

To remove a lesson from the computer:

1. Get to the main AUTHOR MODE message

2. Type "DELETE" and press key NEXT (a message will request the name of the lesson)

3. Type the name of the lesson to be deleted and press key NEXT. Deletion is complete when the AUTHOR MODE display appears. Deletion may be verified by examination of the AUTHOR MODE display.

If a lesson is deleted while students are still working on it, the students' records are automatically updated and the WELCOME TO PLATO message appears at the stations in use by those students. Thus, DELETE provides an alternate method of halting a class session if student data are not being collected. Use of the DELETE option has no effect on authors or students working on lessons other than those deleted. Only one instructor may DELETE at a time. If an instructor attempts to use DELETE while another instructor is using DELETE (or certain other options such as READIN) a message to that effect occurs. The author may return to the main AUTHOR MODE message by pressing key NEXT and attempt to DELETE again after a short wait.

7. The PRINT Option. Printed copies of portions of a lesson or the entire lesson may be produced for authors by means of the PRINT option.

To obtain a printed copy of a lesson

1. Get to the main AUTHOR MODE display
2. Type "PRINT" and press key NEXT. (The PRINT option message will appear).
3. Type the name of the lesson desired
4. If selected sections of the lesson are desired use the ARROW key to move the arrow to that portion of the message and enter the beginning and ending Units of each segment desired.
5. Press key NEXT (a message will indicate that printing is being done).
6. The main AUTHOR MODE display will reappear when the printing has been completed.

Only one author may PRINT, READIN, or DELETE at a time. Thus one author may not DELETE a lesson during the time another author is using PRINT. If someono else is using any of these options the author will receive a message telling him of the conflict. The author thus thwarted should press key -NEXT- to return to the main AUTHOR MODE message and repeat his attempted PRINT, READIN, or DELETE after a short wait. Do not use PRINT near the first of a class session since this may delay the start of classes needing to READIN lessons.

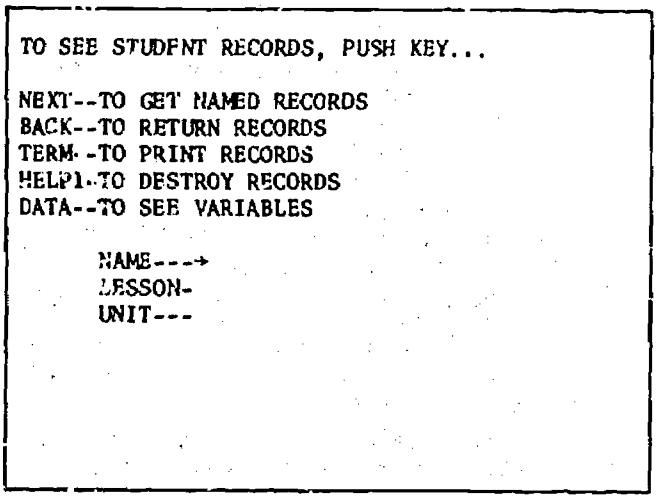
Since printed copies of lessons are produced only at the computer site,authors at remote sites will not have immediate access to their requested print-out. Such authors will recognize the usefulness of having complete author identification contained in the first Unit of each lesson.

8. The RECORDS Option. The final option to be described in this chapter permits the author to specify who will be allowed to use a lesson. Earlier chapters show how a student "signs-in" for a class period. When a student types his name, the disk storage is searched for a "STUDENT RECORD" with the same name. If the proper name is found and the lesson is available, the student may proceed. Otherwise he is stopped. The procedure is analogous to a teacher checking a roll book on the first day of a new semester. Students who show up in the wrong class are directed elsewhere.

The analogy of STUDENT RECORDs to a teacher's roll book may be extended further. Just as a roll book contains space for added information such as grades and attendance, each STUDENT RECORD contains 63 spaces known as STUDENT VARIABLES for recording temporary or permanent information concerning a particular student.

When a student signs in for a lesson his STUDENT RECORD is brought from disk storage and placed in the computer. When a student finishes a class session and signs out (by the TERM - "FINISHED" -NEXT procedure) or is signed out by the instructor (by use of STOP or DELETE), the up-to-date STUDENT RECORD replaces the old record in disk storage. If the student does not get signed out, his old record is not altered. The RECORDS option allows the author to begin new student records or destroy old ones and to move a copy of a student record from disk to computer in order to alter, print, or observe entries in the record.

One of the most useful parts of a STUDENT RECORD is the position of the student in the lesson that he is working in. As the student proceeds through the lesson, a record of his current base Unit is maintained. Thus a student may sign out in the midst of a lesson and return to complete that lesson at a later time. Figure 9.15 shows the RECORDS display which is seen when the RECORDS option is activated. As you can see, there are spaces for the name of a student, the title of a Lesson, and the name of a UNIT within that lesson. These may be typed, erased, or altered by appropriate use of the ARROW, ERASE, and regular keys of the keyboard. Five sub-options are listed on the RECORDS display. A basic sub-option involves getting a copy of a student record from disk storage to the computer. Once the copy is present in the computer, it may be examined and altered by the author at his PLATO station.



```
TO SEE STUDENT RECORDS, PUSH KEY...

NEXT--TO GET NAMED RECORDS
BACK--TO RETURN RECORDS
TERM--TO PRINT RECORDS
HELP1-TO DESTROY RECORDS
DATA--TO SEE VARIABLES

      NAME---→
      LESSON-
      UNIT---
```

Figure 9.15
RECORDS Display

To Get a Student Record:

1. Get to the main AUTHOR MODE display.
2. Type "RECORDS" and press NEXT (RECORDS display will appear).
3. Type the name of the desired student (no more than 8 letters) and press NEXT (Student records for the named student will be sent from disk storage to the computer).

The author can request that a printed copy of the student record be provided. If a student record is altered, the revised copy can be returned to disk storage where it will replace the old version of that particular studen record. When a course is completed, old student records may be destroyed so as to give space on the disk pack for other students. The sub-options will now be explained in more detail.

1. To get RECORDS from the disk pack: Type the name of the student (limited to 8 letters, spaces, punctuation marks or numbers) and press key NEXT. If the student has records assigned, his current Lesson and Unit will appear in those spaces. If the student does not have records assigned, a series of small wedge symbols will appear on the Lesson and Unit lines of the display.

2. To send RECORDS from the computer to the disk pack: When key BACK is pressed, the Name line is checked. If a name is present the copy of RECORDS then in the computer is sent to the disk pack. The RECORDS being placed on the disk pack will replace any already present which are assigned to the same name. If no records having the same name are present, a new RECORD is started. NOTE--if BACK is pressed when no name is present on the Name line, the station is returned to the main AUTHOR MODE display.

3. To print a copy of RECORDS: Press key TERM to get printed copy of the RECORDS which are currently in the computer.

4. To destroy RECORDS: Type the name of the student whose RECORDS are to be destroyed. Press the SHIFT and HELP keys simultaneously and the RECORDS will be deleted from disk storage.

5. To examine or alter TUTOR variables for a student: Get the student's RECORDS from disk and press key DATA. A new display will appear. This new display allows the author to specify what format the 63 variables are to be viewed in. After the format or formats (see Chapter 6) have been specified the variables appear sequentially as key NEXT is pressed.

The process of sending RECORDS to the disk storage should be done with caution. For example, a careless author might send RECORDS for a new student to disk storage without first verifying that no RECORDS for another student with the same name are present. If duplication of names occurred, the older RECORDS would be lost when the new RECORDS replaced them. The following procedure should always be followed in setting up RECORDS for a new class.

To set up RECORDS for a class:

1. Verify that no RECORDS are already present on the disk pack with names identical to those which your students will use. (Check each name on your roll by attempting to get RECORDS from the disk under that name. If the wedge symbols appear, it is safe to use that name; if RECORDS are present, try variations of the name until an acceptable form is found).

2. Set up a standard RECORD for the class. (Indicate the Lesson and Unit names and set any TUTOR variables which the lesson assumes to be preset. If students are to begin at the first Unit of the lesson, the Unit line may be left blank).

3. Send class RECORDS to disk (Fill in each student name and press BACK. When the student name disappears, the RECORDS will have been placed on disk. Other names may be typed in and sent to disk. Note that once a standard RECORD has been set up, it need not be altered unless subsequent students are to have different initial RECORDS).

4. Verify that RECORDS are present. (Try to get RECORDS for each student on your list).

Figure 9.16 shows the display which is seen when an author presses key DATA in order to view TUTOR variables for a particular student. The author must specify the format (see Chapter 6) in which the variables are to be interpreted. If all variables are integers the author may simply type "63" opposite the INTEGER entry. He could then press NEXT 63 times and see each of the 63 variables as interpreted in "I" format. If variables use several different formats, the author might arrange them so that all "A" format variables precede all "I" variables which in turn precede all "F" variables (e.g. variables 1-12 use "A" format, 13-24 use "I" format, and 25-63 use "F" format). This would allow him to specify all formats for viewing or printing variables at once. Figure 9.16

```
THERE ARE 63 VARIABLES

SPECIFY HOW YOU WOULD LIKE THEM
DIVIDED UP.  THEY WILL APPEAR
SEQUENTIALLY WHEN YOU PRESS -NEXT-.


     VARIABLES    NUMBER

     WORD         →12

     INTEGER       12

     FLOATING      39
      POINT
```

Figure 9.16

Format Specification Display

shows how the Format Specification Display would be set up for the above example. Authors need not feel constrained to arrange their variables in the order indicatrd for the Format Specification Display. It is possible to enter the Format Specification Display three times and interpret all variables successively under each format. Only the variables which had actually been stored under a specific format would appear to be "correct" when interpreted under that same format.

Summary

CAUTION: Always return a station to STUDENT MODE after using AUTHOR MODE.

TO ENTER AUTHOR MODE: (from a station not in use by a student) Press key TERM, type entry code word

TO RETURN TO STUDENT MODE: Press key BACK one or more times

AUTHOR MODE OPTIONS: Type code word of options and press key NEXT

Code Word	Use
EDIT	To produce or edit a TUTOR lesson
	Sub-options F ("forward")
	B ("back")
	D ("delete")
	I ("insert")
	S ("save")
READIN	To place a lesson in the computer
START	To make lessons in computer available to students
DATA	To collect student data
STOP	To stop data collection and halt all students'
DELETE	To remove lessons from computer
PRINT	To produce a printed copy of a lesson
RECORDS	To examine or alter student records

FINDING LESSON ERRORS

or

"How to Avoid Looking Grim as You Reap What You Have Sown"

From time to time you will find that there is a sharp difference between the way PLATO presents a portion of your newly written material and the way you intended PLATO to present it. On these occasions it is customary to blame

(1) PLATO (the system),

(2) TUTOR (the language), or

(3) the author (you),

in that order. In practice, however, most errors in lesson presentation are due to mistakes on the part of the author. This is not to say that either PLATO or TUTOR is infallible but only that the least thoroughly tested element in the system (your lesson) is the most likely location of an error. Also, malfunctions of either PLATO or TUTOR tend to be catastrophic and thus immediately apparent to all. For these reasons we will assume here that your problem lies in your own material. Fortunately, most mistakes will be obvious and you will have no trouble correcting them. Some, however, will not be so obvious (at least the first time they are encountered) and some of the suggestions given here may be of help.

Where to Look

One of the best sources of clues to errors is provided by error messages from PLATO. An error listing is given on the screen by PLATO whenever a lesson which contains certain faulty commands is read into the computer for student use. If such a listing appears after you have read in your lesson, it means that PLATO found something wrong with the way you wrote the lesson. You may, for example, have misspelled a command or specified a connection (e.g. by a JUMP command) to a UNIT which was not available. Whenever such errors occur, PLATO ignores the entire line containing the error and adds the error to the error listing. If you then use the lesson as a student, the lesson will behave as though the line containing the error was not present. It should be apparent that error listings should not be ignored.

Another error message which may be displayed on a student's screen is the "FATAL LOOP" message. This indicates that TUTOR found more than 1000 commands during one of your lesson's contingencies (e.g. during a UNIT-C, ARROW-C, etc.). Generally this results from cycling through the same set of commands over and over again. If this "looping" was unintended you can prevent this message from recurring by rewriting the portion of your lesson which caused it. Otherwise, you may want to use the LOOP command (which see) to allow a larger number of commands to be encountered by PLATO during a contingency.

More subtle errors result when a proper command is used in the wrong place or left out completely. No error listing will occur (PLATO can't read your mind) but the effects on your lesson will still be evident. Sometimes the only way to find such errors is to locate the UNIT in your lesson in which things last seemed to be working normally and proceed from there line by line. Some of the more common mistakes to look for are;

(1) Are commands legal in the position you used them? (e.g. are you trying to use a command as an ARROW contingency when it was designed only for use as a UNIT contingency?)

(2) Are commands positioned properly for the desired effect? (e.g. if a command is to be an ARROW contingency is it placed between the ARROW command and the first ANS-type command?)

(3) Do all commands that should have tags actually have them? Leaving the tag off of certain commands can produce bizarre effects.

(4) Are there any extraneous blank lines? A blank line after an ANS command will result in rejection of a "correct" answer by PLATO even though what appears to be the same correct answer is produced by pressing the ANS key (actually it is the "correct" answer followed by a blank line).

As you look through the troublesome UNIT you should also think about how each of the commands and tags will affect what PLATO will do.

With a little experience you will find that an undesired effect will often suggest a possible cause. Even without such experience you can frequently narrow down the possibilities. For example, if a lesson "worked" until some additions or corrections were made it is obvious that the new commands and the commands next to them are good beginning points for your search. Remember that the insertion of certain commands (e.g. the ARROW or ANS command) can change the effect of following commands from a UNIT or ARROW contingency to an ARROW or ANS contingency

TUTOR Variable Problems

Lessons which use TUTOR variables should be given special attention. If possible, before attempting to use such lessons on PLATO, you should work through them "by hand" with extreme values to insure that the lesson is prepared for anything a student might do. You will find that such "hand simulation" will eliminate almost all problems before they happen. When problems are found in lessons you should use the disk student records to examine the value of variables at specific points in the program. Remember to update student records, if you are using them, by pressing key TERM and typing "FINISHED" each time before examining them. On errors which require extensive amounts of student operation time you may want to temporarily insert SHOW commands in the lesson or use the PEEK routine described in Chapter 6 to avoid having to shift into and out of Author Mode at many points in the lesson. In some cases, it may be necessary to change JUMP and GOTO commands temporarily into NEXT commands to allow you to go through complex computational routines a step at a time.

In Conclusion

Above all remember that most errors in TUTOR lessons are not of the sort that require a great deal of work to locate. Also, error location is one ability that shows very rapid effects from experience. Once you have tracked down one or two moderately elusive errors your efficiency at this sort of game will noticeably increase.

Chapter 11

APPLICATIONS

Complex lesson segments are seldom written by neophytes. Once written, however, useful complex lesson segments can often be modified to fit other applications by authors who possess only a scant knowledge of TUTOR. This chapter is intended to propagate knowledge about useful lesson segments written by authors skilled in TUTOR and PLATO operations. Although the segments concern specific applications for the sake of a clear presentation, mention is made of how the segments can be generalized for use in other applications. In adapting a lesson segment to other uses, the novice author should list the parts of his application that he wants to differ from those of the given example. Then, it is usually an easy task to find the TUTOR statements which need to be changed. The statement to be altered will often suggest how the new statement should be written.

Each of the following examples begins with an italicized statement describing a desired lesson situation. Then follows a discussion of the problems involved and the method of solution. Finally, a complete TUTOR lesson segment is given which satisfies the desired lesson situation.

One should not be confused by the combined use of ICALC, FCALC, and CALC commands in the examples. In most cases, the CALC command can replace the ICALC and FCALC commands and give the same results. However, the ICALC and FCALC commands are executed about 10 times faster than the CALC command. Authors can use only the CALC command when first writing a lesson segment. Only if noticeable time delays occur should the author consider "speeding up" the operation by substituting ICALC and FCALC commands. Noticeable time delays usually occur when operations require extensive looping.

GRAPHING A FUNCTION

Students are to explore the contributions from the different parts of the quadradic function, $Y=AX^2 + BX + C$, through graphing the function many times while changing the values of A, B, and C.

A first Unit, Unit SETUP, is used to explain the situation to the student and to obtain values for A, B, and C from the student. These values are put into the floating point variables 1, 2, and 3 by the use of STORE statements. If desired, the judge contingencies for Unit SETUP could check that the student values for variables 1, 2, and 3 fall within a range of values meaningful to the problem and to the coordinate system used in the graph.

Another Unit, Unit GRAPH, displays a coordinate system along with the quadratic equation and the student's values for A, B, and C. Variable 5 is initialized at this time and will contain the value of X from 0 to 10 for the calculation of the quadratic function. For each X unit, 15 points on the graph will be generated giving a total of 150 points. Units GRAPH1 and GRAPH2 plot the graph. The first statement in Unit Graph1 calculates the value of Y. Authors desiring to graph other functions merely need to substitute an alternate statement at this point. The next four statements invert the value of Y for graphing and check that Y is within proper screen bounds. The following WHERE statement prepares the PLATO equipment for plotting a dot at the current Y and X coordinate screen position. Unit GRAPH2 increments the screen position of X, the value of X for calculation of the next Y value, and decrements counter 9. Finally a check is made for the completion of the graph using counter 9.

The Y and X screen position values contained in variables 4 and 6 need further explanation. While a full understanding requires an explanation of the operation of the storage tube device, an explanation sufficient for most purposes starts with the information that there are 180 addressable points on the Y-axis of the screen and 256 X-axis points. The top left corner of the screen is addressed by 0,0. Each character of a TUTOR WRITE statement consumes 5 X-axis points while each line consumes 10 Y-axis points. Thus, the X-axis of Unit GRAPH is centered at about 105 on the Y-axis. The calculated Y-value from the quadratic formula must be subtracted from the number 105 to center and invert the graph on the screen. Since the desired zero of the X-axis is four characters in from the left of the screen, the starting screen X-value is 15 (not 20 since the first character position is at 0).

```
UNIT    SETUP
NEXT    GRAPH
WRITE   ONE CAN OBTAIN AN INTUITIVE UNDERSTANDING OF
        THE CONTRIBUTIONS FROM THE DIFFERENT PARTS OF
        THE QUADRADIC FUNCTION

                  Y = AX^2 + BX + C

        THROUGH GRAPHING THE FUNCTION MANY TIMES WHILE
        CHANGING THE VALUES OF A, B, AND C IN AN
        ORDERLY MANNER.

        CHOOSE THE VALUES OF A, B, AND C...(THEY CAN
        BE POSITIVE OR NEGATIVE)

          A =
          B =
          C =

        THE VALUE OF X WILL RANGE FROM 0 TO 10.
ARROW   1409
STORE   F1
ANS
ARROW   1509
STORE   F2
ANS
ARROW   1609
STORE   F3
ANS
C
C
UNIT    GRAPH
NEXT    SETUP
WRITE   GRAPH OF....               A =
            Y = AX^2+BX+C          B =
         80                        C =
         70
         60
         50
         40
         30
         20
         10
           0 1 2 3 4 5 6 7 8 9 10
        -10
        -20
        -30
        -40
        -50
        -60
        -70
WHERE   125
SHOW    F1
```

```
WHERE   225
SHOW    F2
WHERE   325
SHOW    F3
FCALC   F5=0
ICALC   I6=15
FCALC   F7=1/15
ICALC   I9=149
GOTO    GRAPH1
C
C
UNIT    GRAPH1
CALC    I4=F1*F5*F5+F2*F5+F3
ICALC   I4= 105-I4
GOTO    I4, GRAPH2, X
ICALC   I8 = 180-I4
GOTO    I8, GRAPH2,X
WHERE   I4, I6
PLOT    DOT
GOTO    GRAPH2
CHAR    DOT,4040,4041,4140,4141
C
C
UNIT    GRAPH2
ADD1    I6
FCALC   F5=F5+F7
SUB1    I9
GOTO    I9,X, GRAPH1
```

NUMERICAL ANSWER JUDGING

A student's numerical response (1)must be within a stated range of values and (2)must be associated with a corresponding unit of measurement.

When judging complex numerical answers, the author should attempt to give the student as much help as possible for incorrect answers. This lesson segment gives the student four types of error messages: (1) failure to attach a unit label; (2) improper unit label; (3) numerical answer too big; (4) numerical answer too small.

Unit PHYEQ gives the student two problems correctly answered "14.9 grams" and "802.3 grams". The numerical part of the student's response is put into variable 40. Variable 41 holds the correct answer while variable 42 holds the range of allowable error (thus the student can respond with 14.9±0.02 grams and 802.3±0.5 grams.

When judging, Unit GRAMCK bumps out the numerical part of the response. A check is then made for the failure of the student to include a unit label. Sentence judging is then resorted to in order to separate legal unit labels from unrecognizable labels. If the student's label is "gram" or one of its synonyms, a switch is made to Unit NUMJUDG where the evaluation of the numerical part of the student's answer occurs. If the student's unit label is "decigram," the numerical part of the student's response must be divided by 10 to correspond to the author's answer before going to Unit NUMJUDG. Indeed, using sentence judging separated by RESET commands, a student response in centigrams, milligrams, etc., can be handled.

Unit NUMJUDG checks that the numerical part of the student's answer falls within acceptable bounds. Of importance here is the fact that the FCALC command converts a floating point number into a truncated integer number. Thus, "1.8" becomes the integer "1" (one). Since slight rounding-off errors might occur, the allowable range should slightly exceed the desired range by about .000001% for critical usage.

Once Unit GRAMCK and NUMJUDG are written, they can be used as many times as desired through the use of JOIN commands. In fact, Unit NUMJUDG can be used alone for responses not requiring a unit label. Unit NUMJUDG can also be altered so that variable 42 contains a percentage error rather than absolute range of error.

```
UNIT    PHYEQ
WRITE   FILL IN THE MISSING PARTS OF THIS PHYSICS
        PROBLEM...
        ...
ARROW   823
STORE   F40
FCALC   F41=14.9
FCALC   F42=0.02
ANS     14.9 GRAMS
JOIN    GRAMCK
ARROW   1323
STORE   F40
FCALC   F41=802.3
FCALC   F42=0.5
ANS     802.3 GRAMS
JOIN    GRAMCK
C
UNIT    GRAMCK
BUMP    1234567890.+-
WRONG
WHERE   1601
WRITE   YOU MUST LABEL YOUR ANSWER WITH A UNIT OF
        MEASUREMENT.
MUST    G,GM,GS,GMS,GRAM,GRAMS
GOTO    NUMJUDG
RESET
BUMP    1234567890.+-
MUST    DG,DGM,DGS,DECIGRAM,DECIGRAMS
FCALC   F40=F40/10
GOTO    NUMJUDG
RESET
BUMP    1234567890.+-
MUST    CG,CGM,CGS,CENTIGRAM,CENTIGRAMS
FCALC   F40=F40/100
GOTO    NUMJUDG
RESET
BUMP    1234567890.+-
MUST    KG,KGM,KGS,KILOGRAM,KILOGRAMS
FCALC   F40=F40*1000
GOTO    NUMJUDG
RESET
WRONG
WHERE   1601
WRITE   I DO NOT UNDERSTAND YOUR UNITS.
C
C
UNIT    NUMJUDG
FCALC   F50=F41-F40
FCALC   I51=F50+F42
GOTO    I51,HIGHAN,X
FCALC   I51=F42-F50
GOTO    I51,LOWAN,X
C
C
```

```
UNIT    LOWAN
WHERE   1601
WRITE   YOUR NUMERICAL ANSWER IS TOO SMALL.
JUDGE   NO
C
C
UNIT    HIGHAN
WHERE   1601
WRITE   YOUR NUMERICAL ANSWER IS TOO BIG.
JUDGE   NO
```

REPETITIOUS EXCERCISES

An English to French translation drill consisting of 27 problems is to be presented to the student. The problems are to appear in a random order.

The solution to this programming problem consists in separating the variable parts of the lesson sequence from the constant parts. A Unit can then be constructed which contains the constant parts and which joins in the "proper" variable parts. Such a Unit is called a DRIVE Unit.

Unit INIT explains the situation to the student. The BASE command is present to make Unit INIT a base Unit regardless of how the student reached it. The IPERM statement initializes the problem set to 27 problems. Later use of a RANDP statement will withdraw numbers from these 27 without replacement.

Unit DRIVE consists of several parts: (1) a part setting variable 1 which is used to join in a particular problem, (2) the constant display, and (3) the part joining in the particular problem and answer. The RANDP statement places the next number from the original 27 into variable 1. The following JUMP statement checks whether all the numbers have been withdrawn--a condition indicated when variable 1 is set to zero by the RANDP command. Then follow statements which set up the constant part of the display. Finally, a JOIN statement using variable 1 joins in the particular problem and answer. The WRITE statement in Unit T1 occurs as an ARROW-contingency. While only one ANS statement is given in Unit T1, many can occur along with WRONG statements and comments. Units T2 through T27 are constructed in a similar manner.

By using drive Units, much time and effort can be saved by authors since the "constant" parts of a lesson exercise need be written only once. Many additional features can be easily added to the drive Unit. For example, a TIME statement can be added before the ARROW command to add timing to the drill. A command structure such as:

```
        .

        .

        ICALC         I2=0
        ARROW         1101
        WHERE         801
        JOIN          I1,X,X,T1,...
        JOIN          I2,X,OK
        WRONG
        SUB1          I2
        .

        .

        UNIT          OK
        ADD1          I54
        .

        .
```

will automatically tally in variable 54 the number of times the student answered a problem correctly on his first try. (Remember to initialize variable 54 to zero in Unit INIT). Later in Unit DONE, variable 54 can be shown to the student or used to make a decision as to whether the student should be automatically forced to repeat the drill.

The student may be moved through the problem set in many ways other than by use of the RANDP command. The simplest is to use an ADD1 statement along with a proper ending check. A more complicated method is to continue in the problem set choosing problems randomly by use of the RANDU command until some criterion is met. In any case, let the lesson material dictate the progression through the problem set and the terminator.

```
UNIT    INIT
BASE
WRITE   FOLLOWING IS AN ENGLISH TO FRENCH SENTENCE
        TRANSLATION DRILL.
IPERM   27
C
C
UNIT    DRIVE
NEXT    DRIVE
RANDP   I1
JUMP    I1,DONE,DONE,X
WHERE   501
WRITE   TRANSLATE THE FOLLOWING SENTENCE INTO FRENCH.
ARROW   1101
WHERE   801
JOIN    I1,X,X,T1,T2,T3,...
C
C
UNIT    T1
WRITE   WHAT IS THE CAPITAL OF FRANCE?
ANS     QUELLE EST LA CAPITALE DE LA FRANCE?
UNIT    T2
.
.
.
UNIT    DONE
BACK    INIT
WRITE   IF YOU WOULD LIKE TO TRY THE DRILL AGAIN, PRESS
        KEY BACK. OTHERWISE...
.
.
.
```

ANIMATION USING SLIDES

A sequence of 15 pictures are to show how atoms may collide and form a molecule.

Animation using slides on the PLATO system is limited by three factors. The first concerns the difficulties of mounting consecutive slides so that they are registered with one another. The second limitation is the maximum number of slides available--122. The third factor concerns the possible one-tenth second delay time resulting from the computer's handling of requests other than the animation sequence. These limitations dictate that the animation be of a course nature (i.e., the "movement" is done in large steps by a few slides). The animation can possess a psychological reality if the author includes directional clues such as air sweeps, dotted past positions, etc. The following Units illustrate a solution to the problem of showing slides 15 through 29 consecutively at a rate of 2 per second.

```
UNIT   INIT
CALC   I1=14
JUMP   MOVIE
UNIT   MOVIE
NEXT   MOVIE
ADD1   I1
CALC   I2=I1-30
JUMP   I2,X,DONE
SLIDE  I1
TIME   3)
C
C
UNIT   DONE
.
.
.
```

ANIMATION USING PLOTTING

A mouse is to run randomly through a maze until it finds the food reward.

Unit SETUP is used to explain the situation to the student. In addition, all specific information concerning the appearance of the maze, the starting position of the mouse, etc. is initialized at this time. If desired, the author can allow the student to set these parameters.

Since the example lesson segment is well commented, only the "tricks" will be further discussed. The movement of the mouse occurs by bouncing back and forth between two ARROW-Contingencies using a TIME command. At the first ARROW-C, the mouse is plotted. When the time is up, the second ARROW-C occurs. This erases the previous mouse and plots another. This process continues until the mouse finds the food.

RANDU commands are used to generate mouse moves to fit a predetermined forward-to-turn ratio and type-of-turn ratio. The author can alter these commands to allow the student to set the ratio in Unit SETUP. In addition, the author could program in additional mouse strategies that the student could choose such as "follow the right wall," "turn into openings," and "remember crucial turning points."

```
UNIT    SETUP
WHERE   501
WRITE   WHEN YOU PRESS THE NEXT KEY, YOU WILL SEE A
        MOUSE RANDOMLY RUN A MAZE.

        PRESS KEY D WHEN YOU ARE DONE WITH MOUSE
        WATCHING.
C
C       SET UP HORIZONTAL LINES USING TWO VARIABLES PER LINE
C
CALC    I10=510
CALC    I11=530
C
CALC    I12=1510
CALC    I13=1530
C
C       THE LAST HORIZONTAL LINE IS IN VARIABLE...
CALC    I8=13
C       SET UP VERTICAL LINES USING TWO VARIABLES PER LINE
C
CALC    I30=510
CALC    I31=1510
C
CALC    I32=516
CALC    I33=1216
C
C
CALC    I34=824
CALC    I35=1524
C
CALC    I36=530
CALC    I37=1530
C
C       THE LAST VERTICAL LINE IS IN VARIABLE...
CALC    I9=37
C
C       SET STARTING POSITION OF MOUSE
CALC    I1=612
C
C       SET DIRECTION THE MOUSE IS HEADING
C       -1=NORTH, 0=EAST, 1=SOUTH, 2=WEST
CALC    I3=1
C
C       SET LOCATION OF FOOD REWARD
CALC    I5=1429
C
C
UNIT    MOUSE
C
C       DRAW THE HORIZONTAL LINES
CALC    I60=10
CALC    I63=I8
JOIN    LINES
C
```

```
C       DRAW THE VERTICAL LINES
CALC    I60=30
CALC    I63=I9
JOIN    LINES
C
C       SHOW THE LOCATION OF THE FOOD REWARD
WHERE   I5
WRITE   F
C
ARROW   1830
JOIN    MOVE
ARROW   1830
JOIN    MOVE
C
C
UNIT    MOVE
C       THE FIRST PART OF THIS UNIT IS AN ARROW CONTINGENCY AND
C       MOVES THE MOUSE.
INHIB   ARROW
C       GENERATE A 3 TO 1 RATIO OF
C       FORWARD TO TURN MOVES FOR THE MOUSE
C
RANDU   I60,4
JOIN    I60,X,X,TURN,FORWARD
C
C       SHOW THE CURRENT LOCATION OF THE MOUSE
WHERE   I1
JOIN    I3,N,E,S,W
C
C       TEST IF MOUSE FOUND THE FOOD
CALC    I60=I1-I5
GOTO    I60, X, THANKS, X
C
C       SET TIMING TO 4 MOVES PER SECOND
TIME    15
C
C       HANDLE ANY KEYS THE STUDENT PUSHES
ANS     D
JUMP    SETUP
ANS
JUDGE   IGNORE
TIME    15
C
C
UNIT    LINES
C       THIS UNIT PLOTS ALL THE LINES
ICALC   I61=I(60)
ADD1    I60
ICALC   I62=I(60)
ADD1    I60
LINE    I61, I62
ICALC   I61=I60-I63
GOTO    I61,LINES,X
C
```

```
C
UNIT    THANKS
WHERE   I5
WRITE      THANK YOU
C
C
UNIT    TURN
C       THESE UNITS TURN THE MOUSE TO THE
C       RIGHT, TURN THE MOUSE TO THE LEFT, OR
C       LEAVE THE MOUSE ALONE WITH A
C       1 TO 1 TO 1 RATIO
C
RANDU   I60,3
CALC    I60=I60-2
CALC    I3=I3+I60
CALC    I60=I3+1
GOTO    I60,TURN1,X
CALC    I60=I3-3
GOTO    I60,X,TURN2
C
UNIT    TURN1
CALC    I3=2
C
UNIT    TURN2
CALC    I3=-1
C
C
UNIT    FORWARD
C       THESE UNITS MOVE THE MOUSE
C       FORWARD IN THE DIRECTION HE IS HEADING
C
CALC    I60=10
GOTO    I3,FN,FE,FS,FW
C
UNIT    FN
CALC    I2=I1-100
GOTO    HORCK
C
UNIT    FE
CALC    I2=I1+1
GOTO    HORCK
C
UNIT    FS
CALC    I2=I1+100
GOTO    HORCK
C
UNIT    FW
CALC    I2=I1-1
GOTO    HORCK
C
UNIT    HORCK
C       THESE ROUTINES WILL NOT ALLOW THE
C       MOUSE TO CROSS HORIZONTAL LINES
C
```

```
ICALC   I61=I2-I(60)
ADD1    I60
GOTO    I61, HORCK1, X
ICALC   I61=I2-I(60)
GOTO    I61,X,X,HORCK1
C
UNIT    HORCK1
ADD1    I60
ICALC   I61=I60-I8
GOTO    I61,HORCK,X
ICALC   I60=30
GOTO    VERCK
C
C
UNIT    VERCK
C       THESE ROUTINES WILL NOT ALLOW THE
C       MOUSE TO CROSS VERTICAL LINES
C
ICALC   I61=I2-I(60)
ADD1    I60
ICALC   I61=I61/100
GOTO    I61,VERCK1, X
ICALC   I61=I2-I(60)
ICALC   I62=I61/100
GOTO    I62,X,X,VERCK1
ICALC   I62=I62*100
ICALC   I61=I61-I62
GOTO    I61,VERCK1,X,VERCK1
C
UNIT    VERCK1
ADD1    I60
ICALC   I61=I60-I9
GOTO    I61,VERCK,X
C
C       THE MOUSE HAS PASSED ALL CHECKS
CALC    I1=I2
C
C
UNIT    N
PLOT    NMOUSE
CHAR    NMOUSE
        3443,3446,3447,3450,3451,3542,3544,3545
        3546,3547,3550,3551,3552,3641,3642,3643
        3644,3645,3646,3647,3650,3651,3652,3653
        3654,3655,3742,3744,3745,3746,3747,3750
        3751,3752,3756,4043,4046,4047,4050,4051
C
UNIT    E
PLOT    EMOUSE
CHAR    EMOUSE
```

```
        3546,3646,3746,4046,4346,3447,3647
        3747,4047,4147,4247,4447,3050,3150
        3250,3350,3450,3550,3650,3750,4050
        4150,4250,4350,4450,4550 3051,3451
        3551,3651,3751,4051,4151,4251,4451
        3552,3652,3752,4052,4352
C
UNIT    S
PLOT    SMOUSE
CHAR    SMOUSE
        3446,3447,3450,3451,3454,3545
        3546,3547,3550,3551,3552,3553,3555
        3642,3643,3644,3645,3646,3647,3650
        3651,3652,3653,3654,3655,3656,3745
        3746,3747,3750,3751,3752,3753,3755
        4046,4047,4050,4051,4054
UNIT    W
PLOT    WMOUSE
CHAR    WMOUSE
        3246,3546,3646,3746,4046,3147,3347,3447
        3547,3647,3747,4047,4147,3050,3150
        3250,3350,3450,3650,3750,4050,4150,4250
        4350,4450,3151,3351,3451,3551,3651,3751,3550
        4051,4151,3252,3552,3652,3752,4052
```

NAVY

3 Chief of Naval Research
Code 458
Department of the Navy
Washington, D.C. 20360

1 Director
ONR Branch Office
495 Summer Street
Boston, Massachusetts 02210

1 Director
ONR Branch Office
219 South Dearborn Street
Chicago, Illinois 60604

1 Director
ONR Branch Office
1030 East Green Street
Pasadena, California 91101

1 Contract Administrator
Southeastern Area
Office of Naval Research
2110 G Street, N.W.
Washington, D.C. 20037

10 Commanding Officer
Office of Naval Research
Box 39
Fleet Post Office
New York, New York 09510

1 Office of Naval Research
Area Office
207 West Summer Street
New York, New York 10011

1 Office of Naval Research
Area Office
1076 Mission Street
San Francisco, California 94103

6 Director
Naval Research Laboratory
Washington, D.C. 20390
Attn: Technical Information
Division

20 Defense Documentation Center
Cameron Station, Building 5
5010 Duke Street
Alexandria, Virginia 22314

1 Superintendent
Naval Postgraduate School
Monterey, California 93940
Attn: Code 2124

1 Head, Psychology Branch
Neuropsychiatric Service
U. S. Naval Hospital
Oakland, California 94627

1 Commanding Officer
Service School Command
U. S. Naval Training Center
San Diego, California 92133

3 Commanding Officer
Naval Personnel Research Activity
San Diego, California 92152

1 Commanding Officer
Naval Air Technical Training Center
Jacksonville, Florida 32213

1 Officer In Charge
Naval Medical Neuropsychiatric
Research Unit
San Diego, California 92152

1 Dr. James J. Regan
Naval Training Device Center
Orlando, Florida 32813

1 Chief, Aviation Psychology Division
Naval Aerospace Medical Institute
Naval Aerospace Medical Center
Pensacola, Florida 32512

1 Chief, Naval Air Reserve Training
Naval Air Station
Box 1
Glenview, Illinois 60026

1 Chairman
Leadership/Management Committee
Naval Sciences Department
U. S. Naval Academy
Annapolis, Maryland 21402

1 Technical Services Division
National Library of Medicine
8600 Rockville Pike
Bethesda, Maryland 20014

1 Behavioral Sciences Department
Naval Medical Research Institute
National Naval Medical Center
Bethesda, Maryland 20014
Attn: Dr. W.W. Haythorn, Director

1 Commanding Officer
Naval Medical Field Research Laboratory
Camp Lejeune, North Carolina 28542

1 Director
Aerospace Crew Equipment Department
Naval Air Development Center, Johnsville
Warminster, Pennsylvania 18974

1 Chief, Naval Air Technical Training
Naval Air Station
Memphis, Tennessee 38115

1 Commander
Operational Test and Evaluation Force
U.S. Naval Base
Norfolk, Virginia 23511

1 Office of Civilian Manpower Management
Department of the Navy
Washington, D.C. 20350
Attn: Code 023

1 Chief of Naval Operations, Op-37
Fleet Readiness & Training Division
Department of the Navy
Washington, D.C. 20350

1 Chief of Naval Operations, Op-07TL
Department of the Navy
Washington, D.C. 20350

1 Capt. J.E. Rasmussen, MSC, USN
Chief of Naval Material (MAT 031M)
Room 1323, Main Navy Building
Washington, D.C. 20360

1 Naval Ship Systems Command, Code 03H
Department of the Navy
Main Navy Building
Washington, D.C. 20360

1 Chief
Bureau of Medicine and Surgery
Code 513
Washington, D.C. 20360

9 Technical Library
Bureau of Naval Personnel (Pers-11b)
Department of the Navy
Washington, D.C 20370

3 Director
Personnel Research Laboratory
Washington Navy Yard, Building 200
Washington, D.C. 20390
Attn: Library

1 Commander, Naval Air Systems Command
Navy Department AIR-4133
Washington, D.C. 20360

1 Commandant of the Marine Corps
Headquarters, U. S. Marine Corps
Code A01B
Washington, D.C. 20380

ARMY

1 Human Resources Research Office
Division #6, Aviation
Post Office Box 428
Fort Rucker, Alabama 36360

1 Human Resources Research Office
Division #3, Recruit Training
Post Office Box 5787
Presidio of Monterey, California
93940
Attn: Library

1 Human Resources Research Office
Division #4, Infantry
Post Office Box 2086
Fort Benning, Georgia 31905

1 Department of the Army
U.S. Army Adjutant General School
Fort Benjamin Harrison, Indiana
46216
Attn: AGCS-EA

1 Director of Research
U.S. Army Armor Human Research Unit
Fort Knox, Kentucky 40121
Attn: Library

1 Dr. George S. Harker
Director, Experimental Psychology
Division
U.S. Army Medical Research Laboratory
Fort Knox, Kentucky 40121

1 Research Analysis Corporation
McLean, Virginia 22101
Attn: Library

1 Human Resources Research Office
Division #5, Air Defense
Post Office Box 6021
Fort Bliss, Texas 79916

1 Human Resources Research Office
Division #1, Systems Operations
300 North Washington Street
Alexandria, Virginia 22314

1 Director
Human Resources Research Office
The George Washington University
300 North Washington Street
Alexandria, Virginia 22314

1 Armed Forces Staff College
Norfolk, Virginia 23511
Attn: Library

1 Chief
Training and Development Division
Office of Civilian Personnel
Department of the Army
Washington, D.C. 20310

1 U. S. Army Behavioral Science Research Laboratory
Washington, D.C. 20315

1 Walter Reed Army Institute of Research
Walter Reed Army Medical Center
Washington, D. C. 20012

1 Behavioral Sciences Division
Office of Chief of Research and Development
Department of the Army
Washington, D.C. 20310

1 Dr. Vincent Cieri
U. S. Army Signal School
CAI Project
Fort Monmouth, New Jersey

AIR FORCE

1 Director
Air University Library
Maxwell Air Force Base
Alabama 36112
Attn: AUL-8110

1 Cadet Registrar (CRE)
U. S. Air Force Academy
Colorado 80840

1 Headquarters, ESD
ESVPT
L.G. Hanscom Field
Bedford, Massachusetts 01731
Attn: Dr. Mayer

1 6570 AMRL (MRHT)
Wright-Patterson Air Force Base
Ohio 45433
Attn: Dr. G. A. Eckstrand

1 Commandant
U.S. Air Force School of Aerospace Medicine
Brooks Air Force Base, Texas 78235
Attn: Aeromedical Library (SMSDL)

1 6570th Personnel Research Laboratory
Aerospace Medical Division
Lackland Air Force Base
San Antonio, Texas 78236

1 AFOSR (SRLB)
1400 Wilson Boulevard
Arlington, Virginia 22209

1 Headquarters, U.S. Air Force
Chief, Analysis Division (AFPDPL)
Washington, D.C. 20330

1 Headquarters, U.S. Air Force
Washington, D. C 20330
Attn: AFPTRTB

1 Headquarters, U.S. Air Force
AFRDDG
Room 1D373, The Pentagon
Washington, D.C. 20330

1 Research Psychologist
SCBB, Headquarters
Air Force Systems Command
Andrews Air Force Base
Washington, D.C. 20331

MISCELLANEOUS

1 Mr. Joseph J. Cowan
Chief, Personnel Research Branch
U.S. Coast Guard Headquarters
PO-1, Station 3-12
1300 E Street, N.W.
Washington, D.C. 20226

1 Director
Defense Atomic Support Agency
Washington, D.C. 20305

1 Executive Officer
American Psychological Association
1200 Seventeenth Street, N.W.
Washington, D.C. 20036

1 Dr. W. A. Bousfield
Department of Psychology
University of Connecticut
Stoors, Connecticut 06268

1 Dr. Lee J. Cronbach
School of Education
Stanford University
Stanford, California 94305

1 Professor L. E. Davis
Graduate School of Business Administration
University of California, Los Angeles
Los Angeles, California 90024

1 Dr. Philip H. DuBois
Department of Psychology
Washington University
Lindell & Skinker Boulevards
St. Louis, Missouri 63130

1 Dr. Jack W. Dunlap
Dunlap and Associates
Darien, Connecticut 06820

1 Professor W. K. Estes
The Rockefeller University
New York, New York 10021

1 Dr. John C. Flanagan
American Institutes for Research
Post Office Box 1113
Palo Alto, California 94302

1 Dr. Frank Friedlander
Division of Organizational Sciences
Case Institute of Technology
Cleveland, Ohio 10900

1 Dr. Robert Glaser
Learning Research and Development Center
University of Pittsburgh
Pittsburgh, Pennsylvania 15213

1 Dr. Bert Green
Department of Psychology
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

1 Dr. J. P. Guilford
University of Southern California
3551 University Avenue
Los Angeles, California 90007

1 Dr. Harold Gulliksen
Department of Psychology
Princeton University
Princeton, New Jersey 0540

1 Dr. M. D. Havron
Human Sciences Research, Inc.
Westgate Industrial Park
7710 Old Springhouse Road
McLean, Virginia 22101

1 Dr. Albert E. Hickey
Entelek, Incorporated
42 Plesant Street
Newburyport, Massachusetts 01950

1 Dr. William A. Hunt
Department of Psychology
Loyola University, Chicago
6525 North Sheridan Road
Chicago, Illinois 60626

1 Dr. Howard H. Kendler
Department of Psychology
University of California
Santa Barbara, California 93106

1 Dr. Robert R. Mackie
Human Factors Research, Inc.
6780 Cortona Drive
Santa Barbara Research Park
Goleta, California 93107

1 Dr. A. B. Nadel
General Learning Corporation
5454 Wisconsin Avenue, N.W.
Washington, D.C. 20015

1 Dr. Slater E. Newman
Department of Psychology
North Carolina State University
Raleigh, North Carolina 27607

1 Dr. C. E. Noble
Department of Psychology
University of Georgia
Athens, Georgia 30601

1 Dr. Henry S. Odbert
National Science Foundation
1800 G Street, N.W.
Washington, D.C. 20550

1 Dr. Harry J. Older
Software Systems, Inc.
5810 Seminary Road
Falls Church, Virginia 22041

1 Dr. Leo J. Postman
Institute of Human Learning
University of California
2241 College Avenue
Berkeley, California 94720

1 Dr. Joseph W. Rigney
Electronics Personnel Research Group
University of Southern California
University Park
Los Angeles, California 90007

1 Dr. Arthur I. Siegel
Applied Psychological Services
Science Center
404 East Lancaster Avenue
Wayne, Pennsylvania 19087

1 Dr. Arthur W. Staats
Department of Psychology
University of Hawaii
Honolulu, Hawaii 96822

1 Dr. Lawrence M. Stolurow
Harvard Computing Center
6 Appian Way
Cambridge, Massachusetts 02138

1 Dr. Donald W. Taylor
Department of Psychology
Yale University
333 Cedar Street
New Haven, Connecticut 06510

1 Dr. Ledyard R. Tucker
Department of Psychology
University of Illinois
Urbana, Illinois 61801

1 Dr. Karl L. Zinn
Center for Research on Learning
and Training
University of Michigan
Ann Arbor, Michigan 48104

1 Dr. James J. Asher
Department of Psychology
San Jose State College
San Jose, California 95114

1 Dr. Albert E. Goss
Department of Psychology
Douglass College, Rutgers
The State University
New Brunswick, New Jersey 08903

1 Mr. Halim Ozkaptan, Chief
Human Factors
Martin Company
Orlando, Florida 32809

1 Dr. Alvin E. Goins, Executive Secretary
Personality and Cognition Research
Review Committee
Behavioral Sciences Research Branch
National Institute of Mental Health
5454 Wisconsin Avenue, Room 10A11
Chevy Chase, Maryland 20203

1 Headquarters USAF (AFPTRD)
Training Devices and Instructional
Technology Division
Washington, D.C. 20330

1 Director
Education and Training Sciences
Department
Naval Medical Research Institute
Building 142
National Naval Medical Center
Bethesda, Maryland 20014

1 Dr. Mats Bjorkman
University of Umea
Department of Psychology
Umea 6, Sweden

1 LCDR J.C. Meredith, USN (Ret.)
Institute of Library Research
University of California, Berkeley
Berkeley, California 94720

1 Executive Secretariat
Interagency Committee on Manpower
Research
Room 515
1738 M Street, N.W.
Washington, D.C. 20036
Attn: Mrs. Ruth Relyea)

1 Dr. Marshall J. Farr
Assistant Director, Engineering
Psychology Program
Office of Naval Research (Code 455)
Washington, D.C. 20360

1 Mr. Joseph B. Blankenheim
NAVELEX 0474
Munitions Building, Rm. 3721
Washington, D.C. 20360

1 Technical Information Exchange
Center for Computer Sciences
and Technology
National Bureau of Standards
Washington, D.C. 20234

1 Technical Library
U. S. Naval Weapons Laboratory
Dahlgren, Virginia 22448

1 Technical Library
Naval Training Device Center
Orlando, Florida 32813

1 Technical Library
Naval Ship Systems Command
Main Navy Building, Rm. 1532
Washington, D.C. 20360

1 Technical Library
Naval Ordnance Station
Indian Head, Maryland 20640

1 Naval Ship Engineering Center
Philadelphia Division
Technical Library
Philadelphia, Pennsylvania 19112

1 Library, Code 0212
Naval Postgraduate School
Monterey, California 93940

1 Technical Reference Library
Naval Medical Research Institute
National Naval Medical Center
Bethesda, Maryland 20014

1 Technical Library
Naval Ordnance Station
Louisville, Kentucky 40214

1 Library
Naval Electronics Laboratory Center
San Diego, California 92152

1 Technical Library
Naval Undersea Warfare Center
3202 E. Foothill Boulevard
Pasadena, California 91107

1 Dr. Russ L. Morgan (MRHT)
Training Research Division
Human Resources Laboratory
Wright-Patterson Air Force Base
Ohio 45433

1 Headquarters, Air Training Command
Randolph Air Force Base, Texas
78148
Attn: ATXTD (Dr. Meyer)

1 Mr. Michael Macdonald-Ross
International Training and Education
Company Limited
ITEC House
29-30 Ely Place
London EC1
ENGLAND

1 Commanding Officer
U. S. Naval Schools Command
Mare Island
Vallejo, California 94592

1 Dr. Don C. Coombs, Assistant Director
ERIC Clearinghouse
Stanford University
Palo Alto, California 94305

1 CDR H. J. Connery, USN
Scientific Advisory Team (Code 71)
Staff, COMASWFORLANT
Norfolk, Virginia 23511

1 ERIC Clearinghouse
Educational Media and Technology
Stanford University
Stanford, California

1 ERIC Clearinghouse
Vocational and Technical Education
Ohio State University
Columbus, Ohio 43212

1 Dr. Benton J. Underwood
Department of Psychology
Northwestern University
Evanston, Illinois 60201

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) University of Illinois, Board of Trustees Computer-based Education Research Laboratory Urbana, Illinois 61801	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

THE TUTOR MANUAL

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

programming manual, copyrighted January, 1969 by U. of Ill Board of Trustees

5. AUTHOR(S) (First name, middle initial, last name)

R.A. Avner
Paul Tenczar

6. REPORT DATE January, 1969	7a. TOTAL NO. OF PAGES 172	7b. NO. OF REFS none

8a. CONTRACT OR GRANT NO. Nonr 3985(08)	9a. ORIGINATOR'S REPORT NUMBER(S) CERL Report X-4
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

DISTRIBUTION OF THIS REPORT IS UNLIMITED.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency Office of Naval Research

13. ABSTRACT

This manual was written to explain the use of the TUTOR logic-building language used with the PLATO system. The logic, TUTOR, was conceived by Paul Tenczar in June, 1967 and is designed to transcend the difficulties of FORTRAN for a computer-based educational system utilizing graphical screen displays.

TUTOR consists of about seventy words or "commands" which can be used in various combinations to produce desired effects. Much lesson writing can be done using less than a dozen of these commands. TUTOR was designed specifically for use by lesson authors lacking prior knowledge of and experience with computers. The language is easy to learn and to use. Normally, authors are able to write parts of useful lessons after a one-hour introduction to TUTOR. The simplicity of TUTOR does not limit its applications. Since TUTOR is a true language, the ultimate complexity and flexibility of TUTOR lessons is limited largely by the ingenuity and experience of lesson authors.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
PLATO						
author language						
logic building						
computer-based education						
computer-assisted instruction						
instructional language						

DD FORM 1473 (BACK)

S/N 0101-807-6821

Security Classification A-31409