

Sequencing of Units Within a Lesson

5

We have discussed many units which make different kinds of displays. In some cases, the main units had other units attached to them by means of -do-. Upon completion of a main unit, the student can proceed to the next one by pressing NEXT. A greater variety of inter-unit connections is needed to build a complete lesson which includes optional help sequences, branches to remedial sections when the student is having trouble, an index that gives the student some control over the order of presentation, etc. This section will discuss, in more detail, how to build rich interconnections into a lesson. This discussion builds on the introduction to such matters presented in Chapter 1.

It is often desirable to skip over some units, particularly if they are used as subroutines, not as main presentation units. We have seen that this can be done by using a -next- command to name the main unit which is to follow. For example:

```
unit   one
next  two
do    dispone
at    1515
write This is unit one.
*
unit   dispone
calc  radius←(x←y←200)-50
```

(Continued on the next page.)

```
do    halfcirc
*
unit  two
at    412
write This is unit two.
```

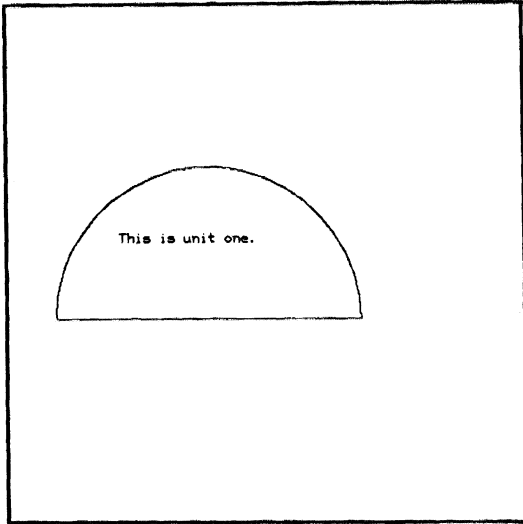


Fig. 5-1a.

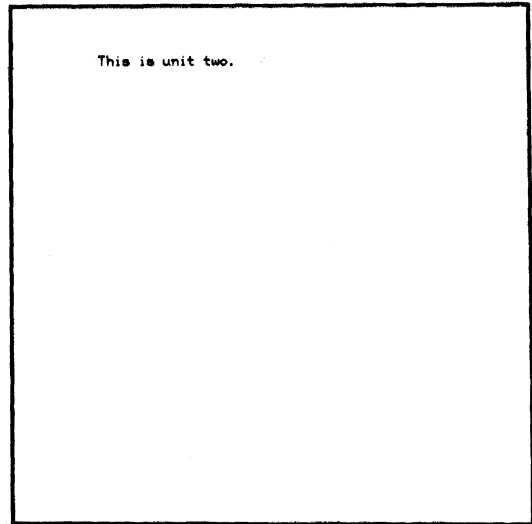


Fig. 5-1b.

When TUTOR begins “executing” the statements in unit “one”, it starts out assuming that the next physical unit, unit “dispone”, will be the next main unit. However, TUTOR encounters a “next two” statement which says, “No, make a note that unit ‘two’ will be next, rather than the next physical unit”. The “do dispone” is then executed, which involves drawing a figure. Finally, we write “This is unit one”, which is at the end of unit “one”. Nothing more will happen until the student presses the NEXT key, at which time TUTOR looks at its notes and finds that unit “two” comes next, whereupon it erases the screen and starts executing unit “two”. Had we not inserted the -next- command, TUTOR would have gone on to unit “dispone” by default.

To put it another way, TUTOR has a pointer which tells which main unit should come next. At the beginning of a main unit, TUTOR places zero in this pointer to indicate that the *next physical unit* should be next.

If no -next- command is encountered, we reach the end of the unit with the pointer still zero, and when the student presses NEXT, TUTOR will by default proceed to the next physical unit. On the other hand, if we encounter a -next- command anywhere in the unit, it will alter this pointer so that later, when the student presses NEXT, the pointer is non-zero and is pointing to whatever unit we have specified.

It should be clear from this discussion that the -next- command can be executed anywhere in the unit without changing its effect. Nevertheless, *it is important to place the -next- command near the beginning of the unit.* The advantage is that you can then see at a glance what is the main sequence flow. If the -next- command is buried far down in the unit, you have to hunt for this crucial information. You put such unit information at the beginning of a unit for the same reason that you define appropriate names for the variables you use: you or a colleague may have to read through the lesson months after it was written!

The following is a simple illustration of how the -next- pointer is handled:

```

unit  silly
next  A
next  B
next  C
*
unit  sillier

```

Well, what unit *will* be next? Answer: unit "C"! The pointer starts out cleared to zero (which implies the next physical unit), then gets set to "A", then to "B", and finally to "C". Each succeeding -next- command overwrites what had previously been in the pointer.

It is also possible to clear the next pointer yourself by -next- with no tag or "next q" ("q" for "*quit* specifying something"). Either of these forms will clear the next pointer so that the next physical unit will come next. In other words, the sequence:

```

unit  start
next  silly
next  q      $$ or just "next" with no tag
*
unit  again

```

will proceed from unit "start" to unit "again" because the "next q" cancels the "next silly".

Such seemingly meaningless manipulations are mentioned here for completeness and as aids to explaining how TUTOR handles a unit pointer, such as that associated with the `-next-` command. These manipulations will make more sense to you later on in the book. The important thing to remember is that you have complete control over the pointer. You can set it or clear it with an appropriate `-next-` command.

The existence of “next q” (and related statements) means that “unit q” is not a permitted statement (you are not allowed to name a unit “q” because of the possible confusion). For similar reasons you will see later that a unit cannot be named “x”.

Another way to utilize pointers is in specifying optional “help” sequences which the student can request by pressing the HELP key. Such optional sequences are important tools in tailoring the lesson to meet the needs of individual students of diverse backgrounds and abilities. Here is an example. (See Figures 5-2a and 5-2b.)

```

unit dipper
help words $$ specify a help unit
at 1215
write Today we will discuss Ursa Major.
*

unit dippy
help words $$ specify a help unit
at 2213
write Ursa Major is in the northern sky.
.
.
.
unit words
at 1525
write Ursa Major is the Latin name for the
constellation which contains
the "Big Dipper".
(Press NEXT for more help,
or Press BACK.)
*
unit words2
at 1525
write "Ursa" means "bear".
"Major" means "bigger".
end

```

SEQUENCING OF UNITS WITHIN A LESSON

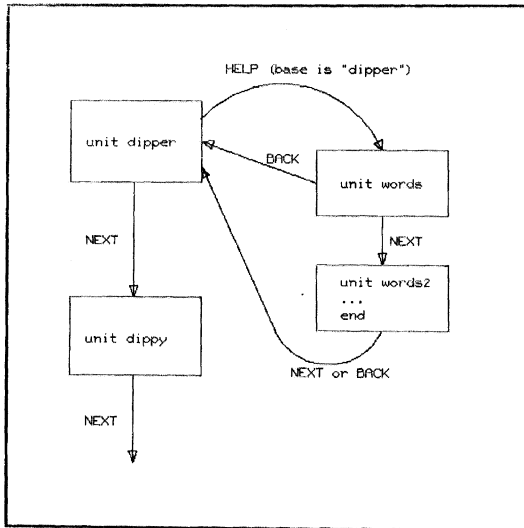


Fig. 5-2a.

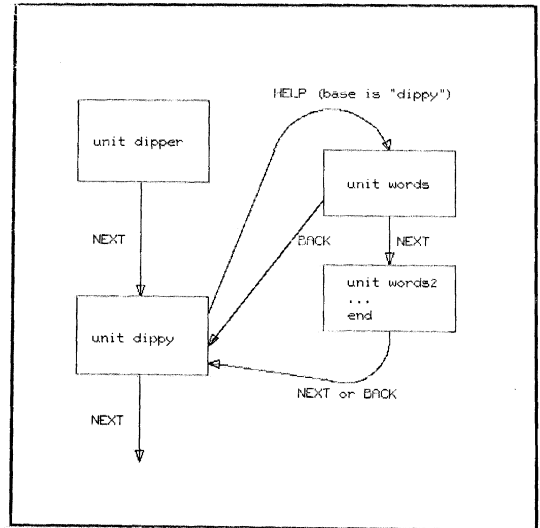


Fig. 5-2b.

The `-help-` command is used to specify a “help” unit, which may be just the first unit in a long help sequence. If you provide help in this way, the student can get it by pressing the HELP key. (Conversely, if there is no `-help-` command, the HELP key has no effect). When the student enters the help sequence, his or her screen is erased to clear the way for the display generated by the first help unit. The student may at any time press BACK or shift-BACK to return to “home base”, the main unit he or she was in when requesting help. A “base” pointer retains the name of the “base unit” (the unit to return to). In the example, if you press HELP in the base unit “dippy”, TUTOR remembers “dippy” and jumps to “words”, from which the BACK key will take you back to “dippy”. If instead you press NEXT, you advance to “words2”, where you can again press BACK or shift-BACK to return to “dippy”. From “words2” you will also return to “dippy” upon pressing NEXT because the `-end-` command in unit “words2” signals the end of the help sequence.

It is almost as though the student had two screens to look at! The student starts the lesson in the first unit of a normal, non-help sequence and advances in this sequence until he or she requests help. At this point, the student turns his or her attention to a different, parallel sequence of units, almost as though that student had turned to use another terminal. The student can get back to the original sequence by pressing BACK, as if he or she had turned back to the original terminal. The usefulness of such a parallel sequence is not limited to help sequences but can be used to

provide review, a desk calculator mode, a dictionary of terms, tables of data, etc., or for any situation in which the student temporarily needs a second terminal “off to the side”.

It is possible to access yet another help sequence when you are already in a help sequence. BACK, however, will return you to the original *base* unit, *not* the help unit you were in when you requested the second help sequence. This is due to the fact that there is only one base pointer, which is not changed by the second help request. If there is already a base unit specification, TUTOR does not alter it.

You can alter the base unit pointer yourself with a -base- command. If you put a -base- command with no tag in unit “words” you will prevent a return to “dipper” or “dippy”. The -base- command with no tag or a “base q” statement clears the base pointer so that TUTOR forgets where to return to and thinks that you are not in a help sequence. (You should notice that the -end- command in unit “words2” is now ignored. The -end- command has no effect in a non-help sequence.) This -base- (blank or “q” tag) is used quite often since it is frequently convenient to put the student into a non-help sequence, even though he reached a certain point by pressing HELP. Also, TUTOR automatically clears the base pointer whenever and by whatever means the student reaches the corresponding base unit.

You can change the base pointer to point to some unit other than the original one. Imagine that we place the following statement in unit “words”:

base dispone

This means TUTOR will eventually return to “dispone” rather than “dipper” or “dippy”. This is occasionally a useful technique. For example, you might like to return to a unit just ahead of the original one in order to ease back into the original context. Notice, too, that while -base- with no tag (or “q”) can change a help sequence into a non-help sequence, so “base unitname” can change a non-help sequence into a help sequence by naming a unit to return to.

You probably will not need all of the features of -help-, -base-, and -end- described above, but hopefully the discussion has clarified how they do their work. You have also discovered some terms which will be quite useful in later discussions and can now talk about “non-help sequences” of “main units” and “help sequences” of “main units”. It should be pointed out that a base unit may have other (auxiliary) units attached to it by -do-; and, of course, you return to the base unit itself, not to one of these attached units, even if the -help- command is located in an attached unit. Moreover, a lesson may be thought of as a collection of

SEQUENCING OF UNITS WITHIN A LESSON

main units which have attached units, and the student moves from one main unit to another. The student may enter a help sequence of main units, each of which may -do- attached units. While the student is in the help sequence, TUTOR remembers which main unit is the “base” unit to return to when -end- is encountered, or when BACK or shift-BACK is pressed. The following is a diagram of this structure:

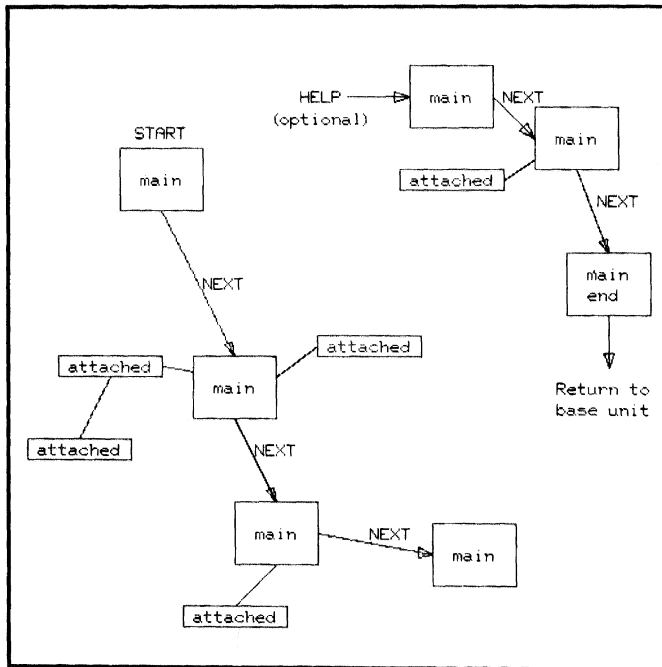


Fig. 5-3.

You may have realized that -help- and -base- are quite similar to -next- in that all three commands set pointers. (The pointers have different uses, however). For example, if we say:

```
unit lotshelp
help a
help b
help c
```

then the last one wins—the help pointer ends up pointing at unit “c”. We saw earlier that -next- works this way. Similarly, “help q” or -help- with no tag will clear the help pointer, thus making the HELP key inoperative.

You may find it helpful to think of a help sequence as a “slow” subroutine. Whereas a -do- command takes us to a unit and right back again, -help- makes possible an optional jump to a unit or to a sequence of units where the student may study for many minutes before returning to the base unit. Aside from the “slowness” and the necessity of pressing keys to go and return, there is one fundamental difference from a -do- situation. In a help sequence, we return from help to the *beginning* of the base unit and re-execute the statements in the unit in order to restore the original display, whereas the return from a -do- is to the statement following the -do-.

This last point is sufficiently important to warrant an example:

```

unit   initial
at     2513
write  Set "a" to 0.
calc   a←0
*

unit   repeat
help   trivial
at     2715
write  Increment "a" to <s,a←a+1>.
*

unit   trivial
at     312
write  Press NEXT or BACK.
end

```

(Of course, “a” must be defined.) If we repeatedly press HELP, then BACK, while we are in unit “repeat” we will repeatedly increment variable “a”. Variable “a” increases by one on every return from the help sequence because the return is to the *beginning* of the base unit, and *all* the statements in unit “repeat” are re-executed. This is necessary to restore (to the screen) the display associated with unit “repeat”, since the entire screen is erased when the HELP and BACK keys are pressed.

This example brings up a fundamental programming point: the question of initialization. We might use a structure like that shown above for counting the number of times the student presses the HELP key (although we would then most likely put the “a←a+1” in the help unit). In order to count something (requests for help, number of wrong answers, etc.), it is necessary to “initialize” the counting variable to zero before starting the process, and this initialization must *precede* (and be outside) the process itself. This can perhaps best be seen by moving the statement “calc a←0” from unit “initial” to the beginning of unit “repeat”:


```

.
.
.
unit   repeat
help   trivial
calc   a←0
at     2715
write  Increment "a" to <s,a←a+1>.
.
.
.

```

Imagine pressing HELP (and BACK) repeatedly. There would never be a change in the displayed value of "a", because on each return from the help unit, "a" is again reset to zero (whereas that was previously done *only* within unit "initial").

The question of initialization will be encountered again and again in various guises. These matters were not mentioned earlier partly because the iterative -do- command had the initialization built-in. For example:

```
do zonk,i←5,13
```

means "initialize 'i' to 5 and do 'zonk', then repeat by incrementing 'i' by one until it reaches 13".

It should be mentioned here that initialization questions are, of course, not unique to programming. The principal and interest due monthly on your car or house loan depend on the *initial* conditions of the loan. When you make fudge, you start with certain ingredients in the mixing bowl (the *initial* condition) and *then* you beat the mixture 200 times. You would no more restart with new, unmixed ingredients after each beating stroke than you would reinitialize a count of student errors after each attempt. In other words, questions of initialization are mainly questions of common sense, and we will make explicit comments about these matters only where confusion is likely. In the case of a return from a help sequence, you might have thought that TUTOR remembers the entire display originally made by the base unit. However, as you have seen, TUTOR must *re-create* the display by *re-executing* the commands in the base unit (which has side effects related to initialization questions).

Now, let's move the "calc a←0" back to unit "initial" and modify the unit to look like this:

```

unit   initial
calc   a←0
jump   repeat  $$ do not wait for the NEXT key
*

```

The `-jump-` command acts much like the student pressing NEXT (the screen is erased and we move to a new main unit). The `-jump-` command is particularly useful in association with initializations, as in this example, where it is necessary to separate initializations from a process in a different unit. It would be superfluous to show the student a blank screen and to make the student press NEXT. Indeed, it should be a basic rule to minimize unnecessary keypresses so as not to frustrate the student. Notice that `-jump-` is *immediate* (like `-do-` and unlike the `-next-` or `-help-` commands) and that statements which follow `-jump-` in a unit *will not be executed* (unlike `-do-`, `-next-`, and `-help-`).

The base pointer is not affected by a `-jump-`. The pointer remains zero if we are not in a help sequence, and it retains its base unit specification if we are in a help sequence. The `-jump-` simply takes us from one new main unit to another without having to press NEXT. Since it starts a new main unit, a `-jump-` cancels any `-do-s` which have been encountered (there will be no return from those `-do-s`).

When moving from one main unit to another, by `-jump-` or by pressing NEXT, the entire screen is erased unless the *first* of these two main units contains an “inhibit erase” statement. For example, the sequence:

```
inhibit erase
jump more
```

will leave the old display on the screen, and displays created by unit “more” will be added to the screen.

Since `-jump-` takes the student from one main unit to another without altering the base pointer, it is possible to take a student to a help sequence immediately without pressing HELP:

```
unit model
.
.
.
base model
jump modhelp
.
.
.
```

Initially, the base pointer is zero because we are in a non-help sequence. Then, a `-base-` command is used to set the base pointer to unit “model” (the main unit we are presently in). The `-jump-` takes us to unit “modhelp”.

Now we are in a help sequence because the base pointer has been set. The return from the help sequence will be to the beginning of unit “model”. Note the difference between “base model” and “base q” in unit “model”: a “base q” statement would clear the already-cleared base pointer, whereas “base model” sets the pointer to “model”.

Summary of Sequencing Commands

You have learned a variety of commands which enable you to control the sequencing of units in a lesson. These include commands which set pointers (-next-, -help-, -base-, etc.) and a couple of immediate branching commands (-do- and -jump-). You have seen how to have two parallel sequences of main units, a non-help sequence and a help sequence, and have used the -end- command to terminate a help sequence. Additional aspects of the connections among units will be discussed in Chapter 6 in the section on the -goto- command. Recall that the LAB, DATA, and BACK keys are activated by -lab-, -data-, and -back- commands, just as the HELP key is activated by the -help- command. The *shifted* HELP, LAB, DATA, NEXT, and BACK keys (abbreviated as HELP1, LAB1, DATA1, NEXT1, and BACK1) are activated by the commands -help1-, -lab1-, -data1-, -next1-, and -back1-. (When in a help sequence, the BACK or BACK1 keys will cause a return to the base unit, unless there are explicit -back- or -back1- commands to alter this.) Here is a unit which uses many of these commands:

unit	central	
help	uhelp	
help1	index	
lab	simulate	
lab1	calc	
data	data	
data1	news	
at	1314	
write	Press	HELP for assistance, shift-HELP for an index, LAB for simulation, shift-LAB for a calculator, DATA for tables of data, shift-DATA for class news.

This is an extreme case, but this unit gives the student *six* choices of help sequences, and which help sequence is entered depends on which key the student presses. In any of these cases, the eventual return will be to this base unit.


The commands -next-, -next1-, -back-, and -back1- are somewhat different in that they do not cause a help sequence to be initiated (pressing the corresponding key does not alter the base pointer, and one simply moves among main units of the help sequence or non-help sequence).

The same conventions apply to all these commands. In particular, a blank tag (or “q”) disables the corresponding key by clearing the associated pointer. A non-help sequence can be changed into a help sequence by specifying a unit to return to with a “base unit” statement. A help sequence becomes a non-help sequence if we encounter a “base q” or “base” statement, since these clear the base pointer.

It is important to point out that *all* the unit pointers, other than “base”, are cleared when we start a new main unit (either by -jump- or by pressing a key such as NEXT, BACK, or HELP). Starting a new main unit, therefore, involves a number of important initializations, including erasing the screen to prepare for the new display (unless there was a preceding “inhibit erase”).

Notice that -jump- and -do- are basically author-controlled branching commands, while -help-, -back-, -data-, etc., permit the student to control the lesson sequence.

There is another way to enter a help sequence, which is particularly useful in offering the student an index to the various parts of the lesson. Suppose the lesson is organized into chapters or topics and you wish to let the student choose his or her own sequence. In particular, the student can skip ahead, go back, or review material. It is desirable that the student be able to go to an index or table of contents at any time. One way to provide access to the index is to put a “data table” statement in every main unit. The student can then hit the DATA key and jump to unit “table” at any time. Unit “table” would contain a list of topics for the student to choose from, and it should contain a “base” statement to insure that the chosen topic be entered as a base sequence. Another way to provide access to this kind of index is by means of a single -term- command:

unit	table
base	
 term	index
at	1218
write	Choose a chapter:
	a) Introduction
	b) Nouns
	c) Pronouns
	d) Verbs

SEQUENCING OF UNITS WITHIN A LESSON

```

arrow  1822
answer a
jump   intro
answer b
jump   unoun
answer c
jump   pron
answer d
jump   verb

```

The presence of "term index" in the unit "table" makes it possible for the student to press the TERM key and type "index" in order to reach unit "table" at any time. (The TERM key is the shifted ANS key on the keyboard.) When the student presses TERM, TUTOR responds by asking the student "what term?" at the bottom of the screen, whereupon the student would type "index". The student then reaches unit "table", where he or she may choose a chapter. You can see that -term- is complementary to -help-. The -help- command in a main unit specifies where to go if HELP is pressed while in that main unit, whereas the presence of -term- in a unit specifies that the unit can be entered from *anywhere* in the lesson. An error is made if another -term- command (with the same tag) is placed in a different unit. In this case, TUTOR would not know which unit to enter.

While the -base- command can be put at the beginning of the unit, there is some advantage to moving it later on in the unit. With -base- commands just before the -jump- commands, the student retains the option of pressing BACK to return to where he or she came from (if he or she doesn't like the available choices). This option is lost if the -base- command has already cleared the base pointer.

The name -term- stems from an early use of this kind of facility to provide a dictionary of "terms", whereby the student has access to the special vocabulary used in a lesson. In such an application, there are as many help units as there are terms to be defined and each unit has an appropriate -term- command:

```

unit  cardinfo
term  cardiac
at    1907
write "cardiac" means "pertaining to the heart".
end

```

When the student types TERM-cardiac, the screen is erased and the definition of "cardiac" is displayed by unit "cardinfo". Immediately

upon pressing NEXT or BACK, the screen is again erased and the student is sent back to the beginning of the base unit. A better procedure in this case would be to change the statement “term cardiac” to “termop cardiac”. The -termop- command refers to “term on page” and permits the display given by unit “cardinfo” to be added to the original display without any erasing.

Except for such dictionary applications, it is strongly recommended that you limit yourself to having only one unit with a -term- in it, and its tag should be “index”. This greatly simplifies the instructions to the student on how to use the lesson and minimizes what he must remember in order to move around in the lesson. In the index unit you describe the various options that are available. Even for providing a dictionary of terms, this scheme is probably preferable (one of the options could be “dictionary of terms”, which in turn would show a list of the words whose definitions are available).

It is possible to have additional -term- commands in the unit to provide synonyms:

```
unit   table
base
term   index
term   contents
term   choice
at     1218
write  Choose a topic . . .
```

These additions insure that the student will reach this unit by TERM-index, or TERM-contents, or TERM-choice.

The -helpop- Command: “Help on Page”

Often the help to be provided when the student presses the HELP key is a brief statement or small drawing which will fit easily on the “page” or screen display which the student is viewing. When this is the case, such help can be added to the screen by means of a -help- command if an “inhibit erase” is used to prevent the current display from disappearing.

A better way is to use a -helpop- command. The statement “helpop hint” specifies that unit “hint” should be done when the student presses the HELP key, without erasing the screen. After going through unit “hint”, TUTOR returns to the point in the lesson where you were waiting for the student to press a key. This could be a -pause- statement, the end of a unit (where you were waiting for the student to press NEXT to

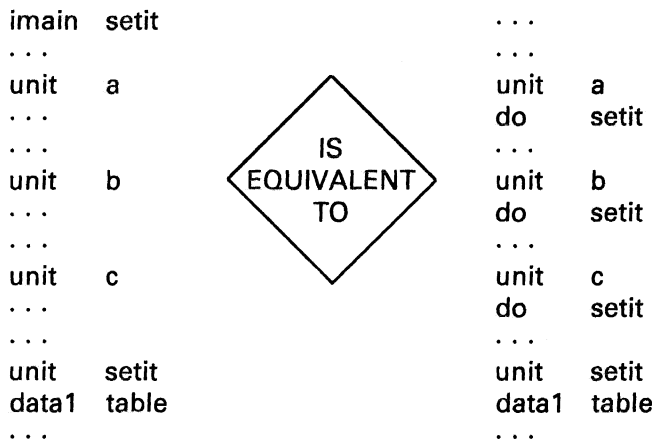
proceed to the next main unit), or an `-arrow-` command where the student was entering a response. The fact that TUTOR returns to the waiting point is an additional advantage of `-helpop-` over the `-help-` command, since return from an ordinary help sequence goes all the way to the *beginning* of the base unit, rather than to the waiting point. (Since the original display is still on the screen when `-helpop-` is used, there is no need to redo the base unit to restore the screen display.) No `-end-` command is needed in a `-helpop-` unit, unlike a `-help-` unit.

The set of on-page commands includes `-helpop-`, `-helplop-` (associated with the HELP1 or shift-HELP key), `-dataop-`, `-datalop-` (for the DATA1 key), `-labop-`, and `-lablop-` (LAB1 key). The `-termop-` command mentioned earlier permits TERM-associated displays “on the page”.

For moving among main units there are the commands `-nextop-`, `-nextlop-`, `-backop-`, and `-backlop-`. These are just like `-next-`, `-nextl-`, `-back-`, and `-backl-`, except that the screen is not erased when proceeding to the named unit. These features can be mixed in one unit. If a unit contains a `-nextop-` command and a `-back-` command, the screen will not be erased when NEXT is pressed, but it will be erased if BACK is pressed.

The `-imain-` Command

An alternative to “TERM-index” is to tell the student to press a key such as shift-DATA to reach an index page. If this index is in unit “table”, you must then put the statement “data1 table” in every main unit, since all unit pointers are cleared when a new main unit is entered. A better way to do this is to use an `-imain-` command which specifies a unit to be done initially in every main unit:



The `-imain-` command names unit “setit” to be done at the beginning of every main unit. This saves you the trouble of placing the statement “do setit” at the beginning of each main unit.

You can specify all kinds of initializations to be performed in each main unit. For example, you might advertise the shift-DATA key with this display at the bottom of the screen:

Press shift-DATA for an index

In this case you would write something like:

```
imain  setit
...
unit   setit
data1  table
at     3218
write  Press shift-DATA for an index
box    3217;3148
```

Now the display will appear with each main unit, and the shift-DATA key will be activated. (Incidentally, if you have blank `-pause-` commands in your units, pressing shift-DATA will merely take the student past the pause, not to the table of contents. Similarly, pressing the TERM key at a blank `-pause-` will not offer TERM capabilities but will merely take the student past the pause. Rather than use a blank `-pause-`, use a statement such as “pause keys=next,data1,term”, as discussed in Chapter 8. With this kind of pause, pressing shift-DATA will take the student to the index, and pressing TERM will give normal TERM features, while pressing NEXT will take the student past the pause. Other keys are ignored.)

The `-imain-` command sets a pointer, just as the `-help-` and `-base-` commands do. You can change the associated unit by executing another `-imain-` command:

```
imain  setit
...
imain  other
```

Notice that the new “imain” unit will *not* be done immediately, but only when a new main unit is entered. You must add the statement “do other” if you want unit “other” to be done immediately. You can stop having an imain-associated unit done by using “imain q”, or “imain” (blank tag), to clear the `-imain-` pointer.

SEQUENCING OF UNITS WITHIN A LESSON

While any key may be used to access an index, many authors have agreed to use shift-DATA in order to provide some uniformity from one lesson to another. This procedure reduces the number of new conventions a student must learn when studying new material.

There is a similar -iarrow- command which can be used to specify a unit to be performed every time a student enters a response. If the -iarrow- command is itself located in the -imain- unit, *all* -arrow-s will be affected.