

Manipulating Data Bases


11

In this chapter we will discuss the tools available in TUTOR for creating and using “data bases” (small or large blocks of data such as test scores, population statistics, map coordinates, etc.). In the process of discussing these tools we will also learn more about the internal workings of the PLATO system.

The -common- Command

The “student variables” v1 through v150 are associated with the individual student. It is possible to use “common variables” which are common to *all* those students studying a particular lesson. These common variables can be used to send messages from one student to another, to hold a bank of data used by all the students, to accumulate statistics on student use of the lesson, to contain test items in a compact, standardized form, etc.

As a first example of the use of the -common- command, let’s count the number of students who have entered our lesson. We will also count how many of these students are female:

```
 common 2 $$ two common variables  
define total=vc1,females=vc2
```

(Continued on the next page.)

```

*
unit      ask
calc      total←total+1
at        1215
write     Are you a female?
arrow     1415
answer    yes
calc      females←females+1
answer    no
no
write     Yes or no, please!
endarrow
at        1615
write     There are <s,total> students, of whom
          <s,females> are female.

```

The `-common-` command tells TUTOR to set up two common variables, `vc1` and `vc2`, which we have defined as “total” and “females”. These common variables are automatically initialized to zero before the first student enters this lesson. The first student increments “total” to one (“`calc total←total+1`”) and may also increment “females”. The second student to enter the lesson causes “total” to increase to two and may also change “females”. Each student is shown the present values of “total” and “females”, which depend on what other students are doing. We must use common variables `vc1` and `vc2` rather than the student variables `v1` and `v2` because the student variables cannot be directly affected by actions of other students. Another way to see this is to point out that when there are five students in this lesson, they share a single `vc1` and a single `vc2`, whereas they each have their own `v1` and their own `v2`: there are five `v1`'s and five `v2`'s but only one `vc1` and `vc2`.

Integer common variables are `nc1`, `nc2`, etc., and indexed common variables are written as `vc(index)` or `nc(index)`.

The statement “`common 2`” tells TUTOR to associate a two-word set of common variables with this lesson. For reference purposes, it is good style to place the `-common-` command near the beginning of the lesson. There can be only one `-common-` statement in a lesson. Like `-define-`, `-vocab-`, and `-list-`, the `-common-` command is not executed for each student. Rather, when TUTOR is preparing the lesson for the first student who has requested it, a set of common variables is associated with the lesson and all these common variables are initialized to zero. Additional students entering the lesson merely share the common variables previously set up.

Suppose a class of fourteen students uses our lesson from 10 a.m. to 11 a.m. The fourteenth student comes at 10:05 and gets a message on the

screen saying "There are 14 students, of whom 8 are female". As long as the lesson is in active use, each new student who enters the lesson increases "total" (vc1). However, when all the students leave at 11:00, the lesson is no longer in active use and will eventually be removed from active status to make room for other lessons. When another class comes at 3:00 p.m., the lesson is not in active use and TUTOR must respond to the first student's request for the lesson by preparing the lesson for active use. In the preparation process the statement "common 2" tells TUTOR to set up two common variables and initialize them to zero. The first student to enter the lesson at 3:00 is told "There are 1 students, of whom 1 are female". She is *not* told "There are 15 students, of whom 9 are female", despite the fact that the previous student (at 10:05 that morning) had been told there were 14 students, 8 female. The "common 2" statement will cause the common variables to be zeroed every time the lesson is prepared for active use.

The type of common which is set up by the statement "common 2" is called a "temporary common". It lasts only as long as the lesson is in active use, and its contents are initialized to zero whenever the lesson is moved from inactive to active status. Temporary common can be used for such things as telling the students how many students are present, what their names are, and whether a student at another terminal who has finished a particular section of the lesson is willing to help a student who is having difficulties. Messages can be sent from one student to another through a temporary common by storing the message in the common area with an identifying number, so that the appropriate student can pick up the message and see it with a -showa-. The lesson simply checks occasionally for the presence of a message.

When a student signs out you usually want to change the temporary common in some way. For example, if you are keeping a count of the number of students presently using the lesson, you increase the count by one when a student signs in and you decrease the count by one when the student leaves. The -finish- command lets you define a unit to be executed when the student presses shift-STOP to sign out:

```

finish  decrease
.
.
unit    decrease
calc    count←count-1

```

In this case unit "decrease" will be done each time a student signs out. Normally the -finish- command should be put in the "ieu". As with -imain-, the pointer set by the -finish- command is not cleared at each new main unit. A later -finish- command overrides an earlier one, and

“finish q” or a blank -finish- statement will clear the pointer. Like all unit pointer commands, -finish- can be conditional. Only a limited amount of processing is permitted in a -finish- unit to insure that the student can sign out promptly.

We can keep a permanent, on-going count of students who enter the lesson by using a “permanent common”. Instead of writing “common 2”, we write “common italian,counts,2”, where “italian” is the name of a permanent lesson storage space and “counts” is the name of a common block stored there. This is the same format used for character sets (the -charset- command) and micro tables (the -micro- command). When the common block is first set up in the lesson space, its variables are initialized to zero. Let’s suppose that the fourteen students who come in at 10:00 a.m. are the very first students ever to use our lesson. The statement “common italian,counts,2” will cause TUTOR to fetch this (zeroed) common block from permanent storage. As before, the fourteenth student arrives at 10:05 and is told “There are 14 students, of whom 8 are female”. At 11:00 a.m. these students leave and our lesson is no longer in active use. At some point, room is needed for other active lessons (and commons), at which point our permanent common, with its numerical contents of 14 (students) and 8 (females) *is sent back to permanent storage*. At 3:00 p.m. the first student (a female) of the afternoon class causes TUTOR to prepare the lesson and retrieve the permanent common from permanent storage *without* initializing the common variables to zero. The result is that she gets the message “There are 15 students, of whom 9 are female”. (There is an -initial- command which can be used to define a unit to be executed when the first student references the common. This makes it possible to perform initializations on a permanent common.)

The key feature of permanent common is that it is retrieved from storage when needed and *returned* in its altered state to permanent storage when the associated lesson is no longer active. In our case, we could enter the lesson months after its initial use and see the total number of students who have entered the lesson during those months. Other uses of permanent common include the storage of data bases accessed by the students, such as census data in a sociology course or cumulative statistical data on student performance in the course.

The Swapping Process

Before discussing additional applications of common variables, it is useful to describe the “swapping” process by which a single computer can appear to interact with hundreds of students simultaneously. The

computer actually handles students one at a time but processes one student and shifts to another so rapidly that the students seem to be serviced simultaneously. In order to process a student, the student's lesson and individual status (including the variables v1 through v150) must be brought into the "central memory" of the computer. After a few thousandths of a second of processing, the student's modified status is transferred out of the central memory (to be used again at a later time) and another student's lesson and status are transferred into central memory. This process of transferring back and forth is called "swapping," and the large storage area where the lessons and status banks are held is called the "swapping memory." The swapping memory must be large enough to hold all the status banks and lessons which are in *active* use; that is, in use by students presently working at terminals. It is not necessary for the swapping memory to also hold the many lessons not presently in use nor the status banks for the many students not using the computer at that time. These inactive lessons and status banks are kept in a still larger "permanent storage" area. (See Fig. 11-1.)

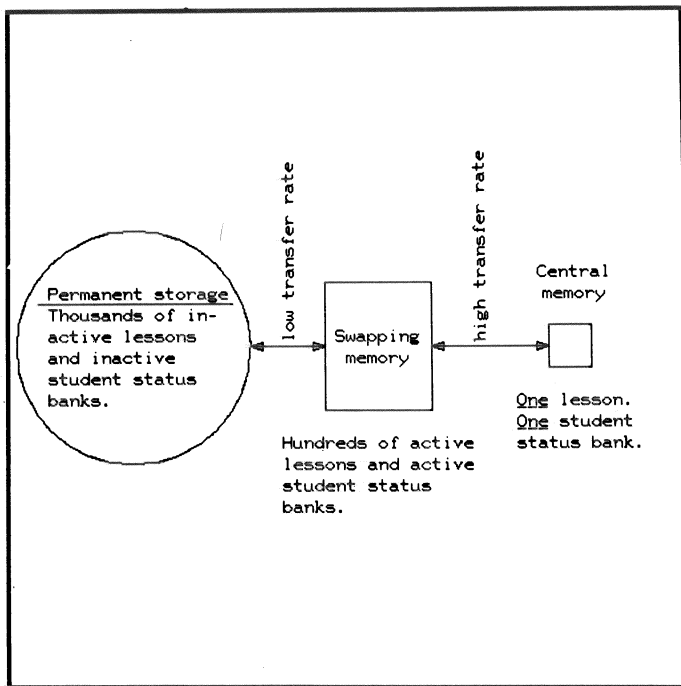


Fig. 11-1.

When a student sits down at a terminal and identifies herself as “Jane Jones” registered in “french2a”, her status bank is fetched from permanent storage to see what lesson she was working on and where in the lesson she left off last time. If the lesson is already in the swapping memory (due to active use by other students), Jane Jones is simply connected up to that lesson, and, as she works through the lesson, her lesson and her changing status bank will be continually swapped to central memory. If, on the other hand, the required lesson is not presently in active use, it must be moved from permanent storage to the swapping memory. (This involves a translation of the TUTOR statements into a form which the computer can process later at high speed.) This fetching of the inactive lesson from permanent storage to prepare an active version in the swapping memory will typically be done once in a half-hour or more often as the student moves from one lesson to another. In contrast, the swapping of the active lesson to central memory happens every few seconds as the student interacts with the lesson. Therefore, the swapping transfer rate must be very high (whereas a low transfer rate between permanent storage and the swapping memory is adequate).

When Jane Jones leaves for the day, her status bank is transferred from the swapping memory to permanent storage. This makes it possible for her to come back the next day and restart where she left off.

The question arises as to why there are three different memories: central memory, swapping memory, and permanent storage. For example, why not keep everything in the central memory where students can be processed? It turns out that central memory is extremely expensive, but permanent storage in the form of rotating magnetic disks is very cheap. Why not do swapping directly between permanent storage and central memory? The rate at which lessons can be fetched from permanent storage is much too slow to keep the computer busy: the computer would handle only a small number of students because a lot of time would be wasted waiting for one student to be swapped for another. If the cost of the computer were shared by a small number of students, the cost would be prohibitively high. In order to boost the productivity of the computer, a special swapping memory is used which permits rapid swapping. This minimizes unproductive waiting time and raises the number of students that can be handled. The swapping memory is cheaper than central memory but considerably more expensive than permanent storage.

There is, therefore, a hierarchy of memories forced on us by economic and technological constraints. The expensive, small central memory is the place where actual processing occurs, and there is never more than one student in the central memory. Material is swapped back and forth to a large medium-cost swapping memory whose most important feature is a very high transfer rate to central memory. Permanent storage is an even larger and cheaper medium for holding the entire set of

lessons and student status banks. It has a low transfer rate to the swapping memory.

Common Variables and the Swapping Process

Now it is possible to describe more precisely the effect of a -common- statement in a lesson. Just as an individual student's lesson and status bank (including the student variables v1 through v1500) are swapped between central memory and the swapping memory, so a set of common variables associated with the lesson is swapped between central memory and the swapping memory. There is in central memory an array of 1500 variables, called vc1 through vc1500, into and out of which a set of common variables is swapped. As long as the -common- statement specifies a set of no more than 1500 common variables, this set will automatically swap into and out of the central memory array vc1 to vc1500. (See Fig. 11-2.) (There is a -comload- command which can be used to specify which portions of a common to swap if the common contains more than the 1500 variables which will fit into central memory.) All 1500 variables in the central memory array are set to zero before bringing a lesson, status bank, and common into central memory, so that any of these variables not loaded by the common will be zero.

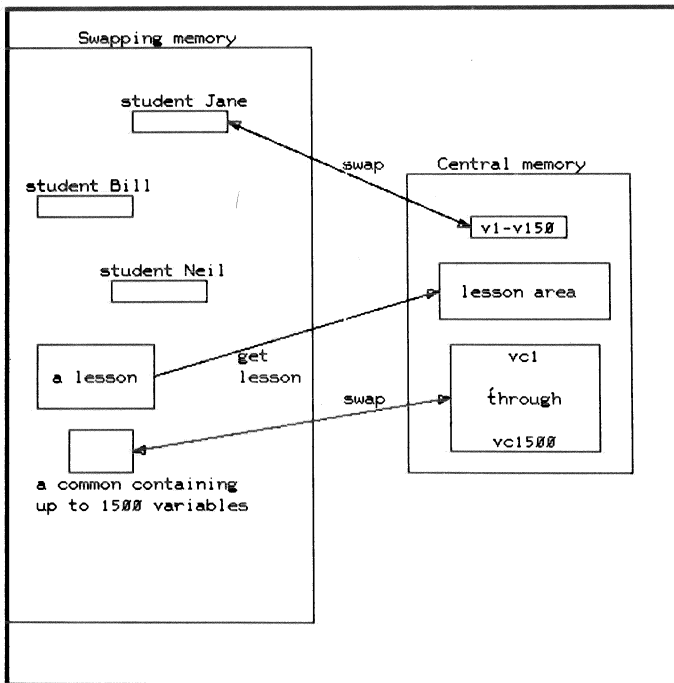


Fig. 11-2.

Note that the student status banks and commons are swapped *in and out* of central memory in order to retain any changes made during the processing in central memory. On the other hand, lessons are brought into central memory but are *not* sent back since no changes are made to the lesson. (A lesson only has to be copied into but not out of central memory.) The separation of the modifiable status banks and commons from the unchanging lessons makes it possible for a single copy of a lesson to serve many students.

It is dangerous to use vc-variables without a -common- statement or to use vc-variables outside the range loaded by the common (e.g., referring to vc3 when there is a “common 2” statement in the lesson). For example, consider this sequence in a lesson which has no -common-statement:

```
.  
. .  
. .  
calc   vc735←18.34  
pause  2  
show   vc735  
. .  
. .  
. .
```

This will show \emptyset , *not* 18.34. The “pause 2” statement causes this student’s material to be swapped out to the swapping memory for two seconds while many other students are processed. When the student is swapped back into central memory, all the vc-variables are zeroed. As a matter of fact, vc735 may temporarily take on many different values during those two seconds as different students are processed. On the other hand, a “common 800” would insure that v1 through vc800 would be saved in the swapping memory and restored after two seconds, so that the “18.34” stored in vc735 would again be available to be shown (unless it had been changed by a student using the same common who was processed during the two-second wait). Similarly, because the student variables v1 through v150 are part of the swapped student status bank, the sequence:

```
.  
. .  
. .  
calc   v126←3.72  
pause  2
```



```
show v126
```

```
.
.
.
```

will correctly show “3.72”. The contents of the student variables cannot get lost in the swapping process because these variables are saved in the swapping memory and restored to central memory the next time this student is processed.

The fact that common variables are shared by all students studying the lesson is extremely useful but can cause difficulties if you are not careful. Suppose you want to add up the square roots of the absolute values of `vc101` through `vc1000`:

```
.
.
.
calc total←0
doto 8sum,index←101,1000
      total←total+[abs(vc(index))].5
8sum
show total
.
.
.
```

This iterative calculation will take longer than one “time-slice” (the computing time TUTOR gives you before interrupting your processing to service other students). You are swapped out and will be swapped back into central memory later to continue the computation. It might take several time-slices to complete the computation, and in between your time-slices other students are processed. This time-slicing mechanism insures that no one student can monopolize the computer and deny service to others. Suppose two students, Jack and Jill, are studying this lesson and sharing its common. Suppose that Jack has reached the part of the lesson that contains the `-doto-` shown above. If, at the same time, Jill runs through calculations that modify `vc101` through `vc1000`, her modifications will be made during the interruptions in Jack’s processing. The total that Jack calculates will, therefore, be based on changing values and will *not* be the total at a particular instant. Jack calculates a partial total, Jill makes some changes, Jack continues to do more calculations in the `-doto-`, then Jill makes further changes, etc. At the end Jack has a peculiar total made up of partial totals made at different times. Even more drastic things will happen if “total” is itself a common variable: Jill might do “total←0” right in the middle of Jack’s summation!

If it is necessary to get an accurate total at a specific instant, it is necessary to lock out Jill and other students from modifying common until the totaling is complete. This is done by doing a “reserve common” statement before starting Jack’s calculation and a “release common” statement after the calculation is complete. The `-reserve-` command checks to make sure no other student has reserved the common, and then reserves the common. The system variable “zreturn” is set to `-1` if the `-reserve-` was able to get control of the common. Otherwise “zreturn” is set to the station number of the student who had previously reserved the common. Normally, if you can’t get the common, you loop waiting for the other person to do a “release common”:

```
.  
. .  
. .  
      8again  
      reserve common  
      branch zreturn,x,8again
```

Notice that you must reserve and release common for Jill as well as for Jack (doing it for one but not the other will not prevent the other from looking at or changing the common).

Don’t forget the “release common” for a student, or other students will get hung up waiting for the common to be available. When a student who has reserved a common signs out of the lesson, TUTOR automatically releases the common.

Note that a lock is certainly needed if different students are *storing* information into the same area of common. There is often no problem with having different students *reading* information out of the same area of common and no problem when storing information in *different* areas of common. Logical conflicts are most serious when *modifying* the *same* part of common. However, even in this case there are usually no problems. In the example of counting the number of students in the lesson, we simply execute “`vc=vc1+1`”, which cannot cause any problems since all of the modifications are completed in one simple step. (Note, however, that a very complicated `-calc-` statement, particularly one involving multi-element array operations, may take more than one time-slice to be performed.)

The `-storage-` Command

In certain applications 150 individual student variables are not sufficient, even when using segmented variables. It is possible to set up

extra storage of up to 1500 variables to give a total of 1650 variables that are *individual*, not shared in a common. A “storage 350” statement will cause a storage block of 350 variables to be set up in the swapping memory for *each* student who enters the lesson. Like -common-, the -storage- command is not “executed” (it is rather an instruction to TUTOR to set up storage when the student enters the lesson). Like temporary common, the storage variables are zeroed when the storage is set up.

A -transfr- command can be used to move common or storage variables from swapping memory into the student variables or into the “vc” area. Usually, however, common is loaded automatically into the “vc” area. If the common is larger than 1500 variables, a -comload- command must be used to specify which part of this large common is to be swapped into and out of which section of vc1 through vc1500. In the case of -storage-, there is no automatic swapping. Instead, a -stoload- command is used to specify what parts of the storage are to be moved into what area of the “vc” variables. Here is a typical example:

```
common 1000
storage 75
stoload vc1001,1,75
```

The common will be automatically swapped in and out of vc1 through vc1000. The 75 storage variables will be swapped in and out of vc1001 through vc1075. It is good form to define all these matters:

```
define comlong=1000,stlong=75
      /stbegin=vc(comlong+1)
      (etc.)
common comlong
storage stlong
stoload stbegin,1,stlong
.
.
calc stbegin<=37.4
```

While -common- and -storage- are “non-executable” commands, -comload- and -stoload- are executable, so that swapping specifications can be changed during the lesson.

The student’s current variables v1 through v150 are saved with other restart information when he or she signs out. Therefore, when the student signs in the next day, these variables will have the values they had when the student left. Storage variables are *not* saved, however. All storage variables are initialized to zero when the storage block is set up upon

entry into the lesson, as with temporary common. If it is necessary to file away more than the standard 150 student variables, you could split up a common into different pieces for individual students. For example, if you need to save 200 extra variables for no more than 20 students, you could split up a 4000-variable common into 20 pieces each containing 200 variables. An alternative is to use "dataset" operations, which permit you to directly control the transfer of blocks of individual data between the permanent storage (magnetic disks) and the swapping memory.

Using "datasets"

A PLATO "dataset" is a file of records kept in the permanent (magnetic disk) storage. You can write some data out to the 5th record of the dataset, then get it back months later simply by reading back the 5th record of that dataset. Each record is made up of many words, and the record word size is specified at the time the dataset is created. (Currently the minimum record size is 64 words.) One record might, for example, hold exam scores for a particular student.

In order to perform operations on a dataset, you first must execute a `-dataset-` command to tell PLATO which of your datasets you are going to be working on at the moment. You can then execute any number of `-dataout-` commands to send data out to the dataset, and any number of `-datain-` commands to read such information back. You can use a `-reserve-` command to reserve specific records, similar to using a "reserve common". You must use a `-release-` command to permit others again to manipulate those records. (For details, see the PLATO on-line "aids".)

Sorting Lists

When manipulating a data base it is often necessary to sort a list of items into alphabetic or numeric order. The `-sort-` (numeric) and `-sorta-` (alphabetic) commands will transform a disordered list into a sorted list. These commands will also sort an associated list of items at the same time. For example, you might have student names in one part of a common, and corresponding grades in another part of the common. You could use a `-sorta-` command to place the names in alphabetical order, and at the same time you could have the `-sorta-` command similarly re-order the grades to correspond with the altered order of the students. (See the PLATO on-line "aids" for details.)