ED 072 632 EM 010 720

AUTHOR Freed, Michele; Bunderson, C. Victor
TITLE Development of an APL Program for Generating Punctuation and Usage Exercises in Freshman English. Technical Report Number 13.
INSTITUTION Texas Univ., Austin. Computer-Assisted Instruction Lab.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE Dec 71
NOTE 26p.

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *College Freshmen; *Computer Assisted Instruction; *Computer Programs; *English Education; Language; Language Arts; Program Descriptions; *Punctuation

ABSTRACT

During the years 1968-1971, the Computer-Assisted Instruction Laboratory at the University of Texas at Austin has developed and implemented three major program designs for use in teaching English punctuation and usage. The initial design was a frame-by-frame approach in which each instruction was prepared by the author and coded separately. The second program took the parts of the original program that could be generalized (i.e., the sentences from the quizzes, exercises, and examples) and put them into sentence pools that were referenced by the program. The third program utilized the workable concepts of the second program, added to them, elaborated on them, and translated the entire program into APL (programing language). Comparisons among the features of the three versions are made, and an appendix provides a number of flow charts from the programs. (Author/RH)

THE UNIVERSITY OF TEXAS AT AUSTIN

Computer Assisted Instruction Laboratory

AUSTIN

DEVELOPMENT OF AN APL PROGRAM FOR GENERATING PUNCTUATION AND USAGE EXERCISES IN FRESHMAN ENGLISH

Michele Freed and C. Victor Bunderson

Technical Report No. 13

December 1971

Supported by:

THE NATIONAL SCIENCE FOUNDATION
Grant GJ 509 X

The University of Texas at Austin
Computer-Assisted Instruction Laboratory
Austin, Texas 78712

PUNCT COURSE DEVELOPMENT

During the years 1968-1971, the Computer-Assisted Instruction Laboratory at The University of Texas at Austin has developed and implemented three major program designs for use in teaching English punctuation and usage. The initial design was a frame by frame approach in which each instruction was prepared by the author and coded separately. The second program took the parts of the original program that could be generalized, i.e., the sentences from the quizzes, exercises, and examples, and put them into sentence pools that were referenced by the program. The third program has utilized the workable concepts of the second program, added to them, elaborated on them, and translated the entire program into APL.

PUNCT

Course Design

The first program, called PUNCT, was developed and tested under a grant from the McGraw-Hill Publishing Company. PUNCT is an adaptation of the programmed text English Review Manual by James Gowen. For PUNCT the text was reorganized to permit the student to progress through the course at his own rate, skip units which he understands, take instructional sequences for units he does not understand, and request remedial instruction on topics related to instructional sequences. Because the student continually interacts with it, the program is more than an expensive page turner. The computer gives the student constant feedback evaluating his input and determining his path through the course.

Disadvantages

The course is massive and each frame has been individually designed and coded. Because of the method of coding and preparation used for implementation in Coursewriter II, PUNCT is almost impossible to debug completely. For a course designed in this way, the author writes responses for correct, wrong, and unanticipated answers for each frame. A modification such as replacing a sentence requires new coding for the entire sentence--the specification of the anticipated light pen response areas, new text displays, and new response messages. Motivated by these problems, the authoring team adapted PUNCT to a new design, PUNCT2-CW.

PUNCT2-CW

Course Design

This adaptation of PUNCT was designed under a grant from the National Science Foundation. PUNCT2-CW, like PUNCT, is programmed in Coursewriter II for use on the IBM 1500 system with a 1512 cathode ray tube (CRT) and light pen. The format of PUNCT is retained while the hard-coded sentences are replaced by calls to a data base, in this course a pool of sentences. These sentences are divided into groups according to sentence patterns which are called sentence prototypes. Although the logic for the PUNCT2-CW program design is sound in its organization and generalization, most of its advantages are negated by slow system response time.

Disadvantages

The Coursewriter language, even when used by a clever and experienced programmer, is not the powerful and rapid tool needed for string processing such as that demanded for the effective implementation of PUNCT2-CW. On-line debugging of PUNCT2-CW was done only when no students or other course authors were signed on the system because execution of PUNCT2-CW at one terminal slowed the system beyond a reasonable response time for other users. The slowing of the system is attributable to the frequent calling of functions which must be moved in and out of core memory.

The primary goals of the PUNCT2-CW designer were to eliminate hard-coded sentences, and to provide authors with a convenient way to manipulate and add to the sentence pool. PUNCT2-CW was a limited success. The macros designed for PUNCT2-CW were limited in scope and not all of the frames in PUNCT were translatable into the data base design. Instead, in the segments selected for adaptation these unique frames were hard-coded for PUNCT2-CW as well. The author entry system remained complex; the macros had many variables; and intricate branching linked the macros together. (For further explanation of PUNCT2-CW, see National Science Foundation Technical Report 6.) Because of these disadvantages, PUNCT2-CW was altered and translated into APL; this version is called PUNCT2-APL.

PUNCT2-APL

Course Design

When designing PUNCT2-APL, which was also designed under a National Science Foundation grant, the instructional design team utilized and expanded the original data base concept. In this APL adaptation the disadvantages of PUNCT2-CW were eliminated while positive modifications were made. The format of PUNCT2-APL is essentially the same as that of PUNCT2-CW with the most significant variations being the programming language and the type of terminal used. Using APL, the designers were able to create a course with the data base capabilities of PUNCT2-CW and the response time of PUNCT.

COMPARISONS

Language and Terminal Differences

Although it offers a display with versatile response and insert capabilities, the 1500 CRT proves to be limiting. The distance limitation of the CRT terminal from the computer becomes important if mass dissemination is a consideration. Connected by telephone line to the computer, the 360/50 typewriter terminal used for PUNCT2-APL can be placed in locations which are easily accessible to students, such as study facilities in dormitories or terminal centers in various campus locations.

With the CRT the student is able to use the light pen to simulate inserting punctuation in a sentence, but the student's typing the punctuation in the proper position serves the same purpose. In the Coursewriter versions of PUNCT and PUNCT2-CW, the student touches, with the light pen, a word which he wants punctuation to follow; if his response is correct, the punctuation mark is inserted. If his response is incorrect, he is given a message which tells him that his response is incorrect. The response time in PUNCT, because of its straightforward programming, is almost immediate. System response time is so slow on PUNCT2-CW that it prohibits running test students. After entering a light pen response, the student must wait until his answer is processed and evaluated before he can make another response. The APL language used in PUNCT2-APL efficiently handles the processing demanded by the data base and promotes modular programming. The student types in his additions for an entire sentence before he enters his response. The system responds by typing an X under each incorrect punctuation mark inserted by the student and under the position of each omitted punctuation mark. The entire corrected sentence is then typed for the student. Using APL, the designers were able to create a course with the data capabilities of PUNCT2-CW and the response time of PUNCT.

The initial reaction of the designers to using the typewriter terminal for PUNCT2-APL was negative because the student would have to wait for the information to be typed rather than have the rapid presentation of a new screen of information on the CRT. However, the Coursewriter version of PUNCT2 moves so slowly that the PUNCT2-APL student using a typewriter would probably have the new message typed to him before the PUNCT2-CW student using a CRT has his answer processed.

Another disadvantage of PUNCT2-APL is that APL does not have a lower case alphabetic keyboard. This problem is most obvious in units dealing with capitalization. However, once the student adjusts to seeing only upper case letters, a convention can be established for use on exercises dealing with capitalization.

In some exercises the APL terminal has advantages. Several units of PUNCT are designed with a format other than the light pen format discussed above and adapted to PUNCT2-CW. In these units, dealing with quotation marks and apostrophes, the student's responses must be more precise than those obtained by pointing with a light pen. The PUNCT student spaces through the displayed sentence and types punctuation marks in the correct positions. The student must learn conventions for moving the cursor more than one space at a time; if many users are signed on simultaneously, the student might wait several seconds between each move of the cursor. All PUNCT2-APL exercises employ for the response format the technique of spacing through the sentence. The APL terminal has a power space key which eliminates the time delay without sacrificing precision. The student presses the power space and the type ball moves across the line to the position in which the student wants to insert punctuation. This new format permits the easy adaptation of certain PUNCT segments to APL which would not have been readily adapted to the PUNCT2-CW format.

The PUNCT2-CW design was limited so that not all frames in PUNCT were recoded for PUNCT2-CW. Instead all of the generalized frames from several lessons were translated, but many one-of-a-kind frames remained hard coded. By expanding the PUNCT2 capabilities in APL, the programmer has been able to eliminate the remaining hard coding that was necessary in PUNCT2-CW. New functions and new sentence prototypes have been designed so that all sentences can be randomly selected from the sentence pool. An example of a hard-coded frame in PUNCT2-CW which is generated in the APL version is one in which the student is asked to identify certain elements in a sentence. The WHICH function is used to match the student's answer with the name of a sentence unit, and a response is created which identifies the unit in a randomly selected sentence. Modification of the course, addition of new sentences, and debugging can be done more easily in PUNCT2-APL. Flowcharts of eleven representative functions are given in the appendix.

PUNCT2-CW lacks the wrong answer (wa) capabilities assumed in Coursewriter and utilized in PUNCT. The wrong answer logic searched for a match, executed the minor commands associated with the match (such as displaying a message, etc.), and branched back to permit the student to respond again. In PUNCT anticipation of incorrect responses were coded only when the author considered an error to be common. In PUNCT2-CW, there is no way for an author to reference a sentence unit without deleting it from the sentence, so that no messages telling the student that his response was incorrect and why are used with the sentences called from the pool. Instead, every wrong answer elicits a standard response that tells him that his answer is incorrect, but gives him no specific information. An APL function (WHICH) has been designed to generate wrong answer responses. This function takes the name of the unit in which, or the names of the units between which the student has tried to insert punctuation and creates a message telling the student why his entry is incorrect. The effect of the message is that of a specifically written message in PUNCT except that these messages are generated and do not have to be written for each wrong answer.

The PUNCT student sees essentially the same course as the student who takes PUNCT2-CW, but since in the latter the sentences are called from a pool rather than being hard coded, the PUNCT2-CW student sees new sentences in examples, exercises, and tests if he must repeat a sequence. In PUNCT each sentence is coded as it is needed and if the author refers to the sentence later in the program, it must be recoded.

In PUNCT2-CW the sentences are loaded randomly into the appropriate category. The program calls the sentences from the category in order as they are needed. When all sentences of a prototype have been used, the program starts through the sentences again. PUNCT2-APL, on the other hand, generates a list of random numbers corresponding to the numbers of the sentences and uses them in this random order. If all of the sentences are used, a list of numbers is generated again and the sentences are reused in the new random order.

An additional feature of PUNCT2-APL is that the student is given an opportunity to try additional sentences when he makes an error. The author sets a variable (for the present time it is set at five); the student is presented a sentence of a certain type. If he does not make the necessary corrections, he is given the correct answer and another sentence of the same type to try. The procedure is continued until the student answers correctly or has tried five sentences.

Author Entry System

PUNCT is an entirely hard-coded course; each sentence, anticipated response, and response message must be written by the author and coded by a programmer. Therefore, PUNCT is extremely long and difficult to debug completely. Any modification entails extensive changes in the course. Adding drill exercises involves the preparation and coding of each sentence added to the drill and is a long and cumbersome task. PUNCT2-CW is based on a sentence pool, and a limited number of macros are used to call and process sentences. Although there is still occasion for error, the format makes the bulk of the course easier to debug. Drills are much easier to add, but major modifications are difficult to make. In addition to using the data-based program for calling sentences from a pool and processing answers, the author can add new sentences to the pool which suit his purpose and taste without having to specify each anticipated answer and message.

A teacher might want to insert sentences that are related to other topics being studied by his students. With the help of a clerk, he could replace the old sentence pool with sentences more relevant to his class. An author who is satisfied with the sentence pool may change the instruction. A teacher may find that the approach taken in PUNCT2-CW is too traditional, but may find the sentences acceptable. He can write new instructional frames which can be inserted by a programmer. Another teacher may find that neither the instructional sequence nor the sentences in the pool are satisfactory. With orientation and the help of a programmer, the teacher can write a new course using the macros and structure of PUNCT2-CW. However, this last option would entail a great deal of effort on the part of both the teacher and programmer. Nevertheless, the basic design provides a flexibility for the teacher who is willing to devote time to making the course suit his needs.

Great advancements toward a simplified author entry system have been made during the project development. The most dramatic difference in author entry systems is illustrated by the author's drafts of the pretest on dates and addresses from all three courses in Appendices 1, 2, and 3. These drafts indicate that the PUNCT author had to specify all details for the programmer. He wrote each question, correct answer, wrong answer, and unanticipated answer responses as well as the instructional unit. The author was aware of the Coursewriter language and often used code to communicate with the programmer. Very little creativity remained within the programmer's realm of responsibility; he kept score, prepared screen displays, and created macros to relieve the tedium of his work.

The breakdown of tasks was somewhat different in PUNCT2-CW and PUNCT2-APL. Because the course was already written, the author worked with an instructional designer who created the data-based system using the course material of PUNCT. The instructional designer was aided by a skilled programmer who prepared the Coursewriter macros for PUNCT2-CW and the functions for PUNCT2-APL to meet the design specifications. Programmers utilized these macros and functions in course implementation.

Although the author entry system of PUNCT2-CW is less detailed than that of PUNCT, it is not a system that can be used easily and without extensive orientation. The macros used for calling and correcting sentences pulled from the sentence pool have several variables and must be used in conjunction with Coursewriter functions and coding. Branching becomes intricate when corrections are made for a list of sentences as in a pretest. Six macro calls are needed to select and display a sentence, process answers, display the sentence with student corrections, and display it with the corrections that the student has omitted. These macros are described in detail in National Science Foundation Technical Report 6. Both the author and programmer must be aware of macro capabilities. Adding new sentences to the data base requires many steps including changing variables in several macro calls. A sentence is added to the pool by macro <u>sp</u> after it has been processed by subroutine <u>zipzip</u>. <u>Zipzip</u> is executed at a typewriter terminal with a clerk entering the sentence units one at a time. The product of <u>zipzip</u> is a string of numbers indicating the number of units in the sentence, the length of each unit, the length of the last word of each unit, and the sentence with unit markers.

PUNCT2-APL provides a simplified author entry system. One function call replaces six macrc calls and extensive coding in PUNCT2-CW. For example, the statement 3 DO 3 5 determines that a prototype 3 sentence is being called and that sentence units 3 and 5 are to be deleted. This same function processes answers, generates messages, and inserts the correct answer. Subroutine zipzip is unnecessary in PUNCT2-APL since all necessary computation occurs within the function. Instead, sentences are loaded with markers (¨) that indicate the end of sentence units. For example, a typical sentence entry would be:

SENTENCE←'I HAD¨,¨INDEED¨,¨HOPED YOU WOULD ASK THAT QUESTION.¨'

The deletion of sentence units 2 and 4 would result in the presentation of the sentence with commas deleted.

In addition to being easier to work with, the APL functions also eliminate the need for the extensive branching used with the Coursewriter macros. To replace an entire sentence in the sentence pool, one line of code instead of numerous macro calls must be changed. To change a sentence in PUNCT a complete new set of coding must be prepared and inserted in place of the old.

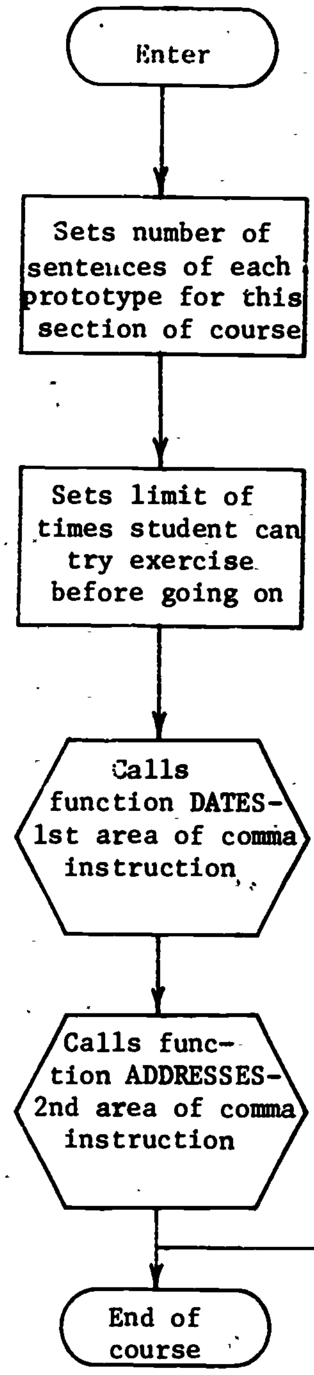
PUNCT2-APL appeared to be a success, but the equipment necessary to test it on a large number of students was not available at the University, so an effort was made to adapt the material once again to the IBM 1500 system. The previous 1500 version had been programmed in Coursewriter II and the language was thought to be the difficulty. The new version was to be in APL. The adaptation was not difficult to make. However, early in the process it became apparent that the system rather than the language was the problem. Once again the system response time became so long that the program was not workable; the translation from 360 to 1500 APL was discontinued. This version, had it proved workable, would have been preferred because it employed the light pen and CRT.

CONCLUSION

The work done on PUNCT2-APL furthers efforts being made to simplify the author entry system. By eliminating the need to specify each detail, the design team has removed an obstacle for the creative author. In addition, PUNCT2-APL demonstrates the effective use of modular programming and a data base for instruction in English. More important than the segments of PUNCT which were adapted to PUNCT2-APL are the concepts that were developed. These concepts can be extended for use in areas of English instruction other than those covered by PUNCT.

APPENDIX

GO


Enter
Sets number of sentences of each prototype for this section of course
Sets limit of times student can try exercise before going on
Calls function DATES-1st area of comma instruction
Calls function ADDRESSES-2nd area of comma instruction
End of course

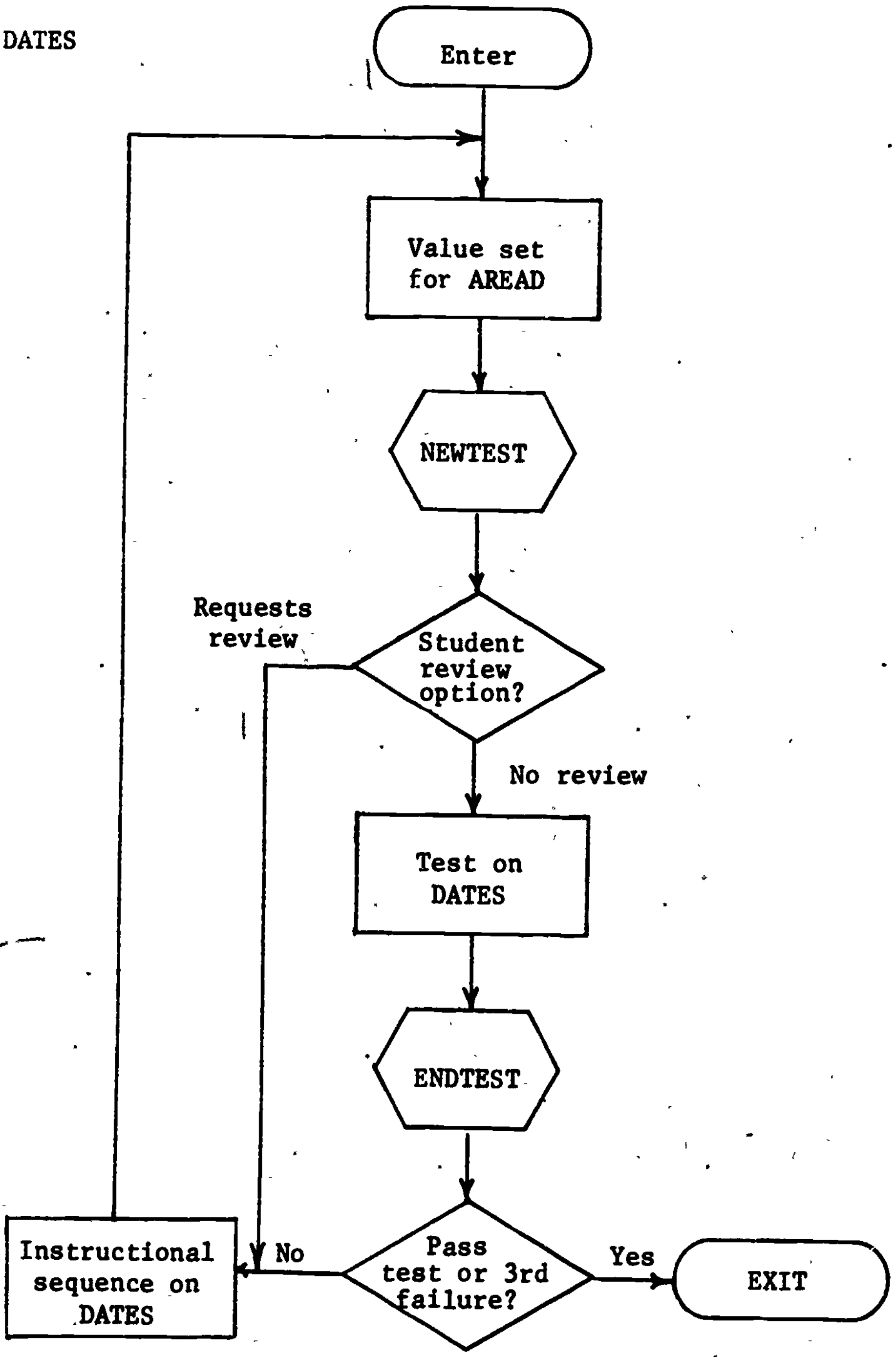
This must be updated as more sentences are added to prototypes.

Specifies name of punctuation being used and loads punctuation mark for messages to student. Clears counters and variables.

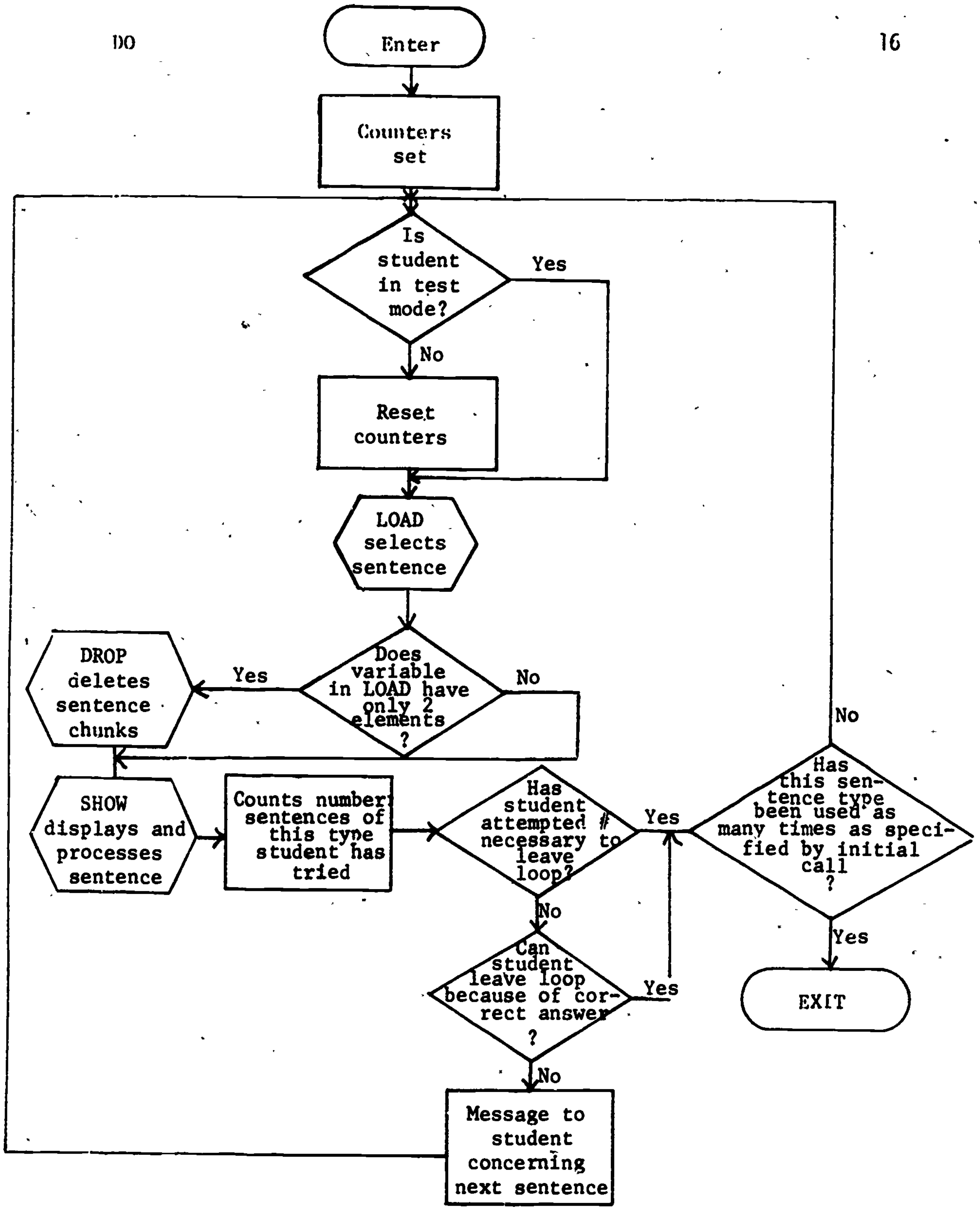
Other units of instruction will be added here.

GO starts the course. It nests all of the other functions. It will be expanded as other units are programmed. There should be other main control segments like this one to provide restart points.

DATES

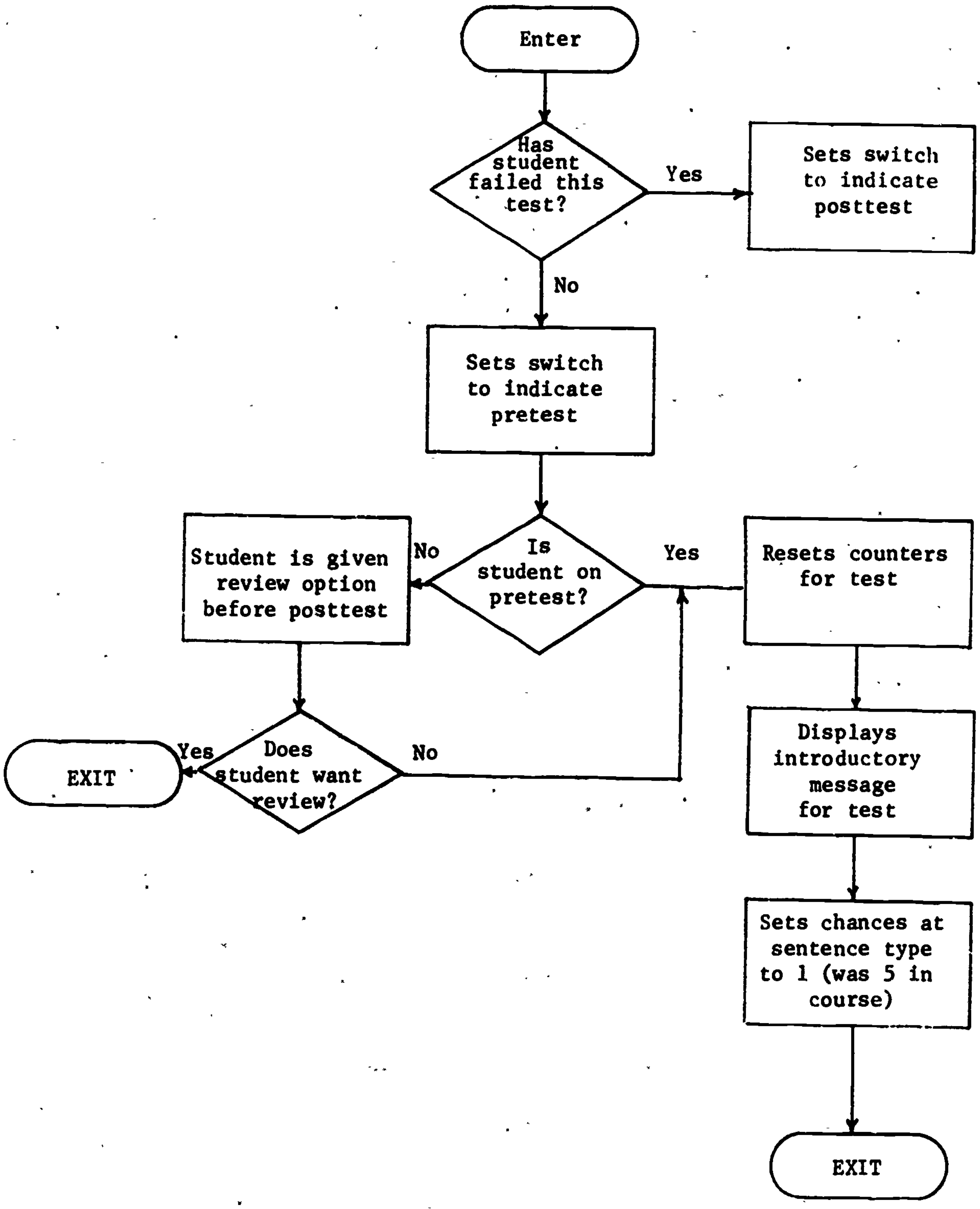


This function has no parameters; it calls an entire unit of instruction including a pretest and posttests. At EXIT, the student has passed the pretest, the posttest, or has been through the instructional unit twice without being able to pass the posttest. The function ADDRESSES flowchart is identical to this one.


Enter
Counters set
Is student in test mode?
Yes
No
Reset counters
LOAD selects sentence
Does variable in LOAD have only 2 elements ?
Yes
No
DROP deletes sentence chunks
SHOW displays and processes sentence
Counts number sentences of this type student has tried
Has student attempted # necessary to leave loop?
Yes
No
Can student leave loop because of correct answer ?
Yes
No
Message to student concerning next sentence
Has this sentence type been used as many times as specified by initial call ?
No
Yes
EXIT

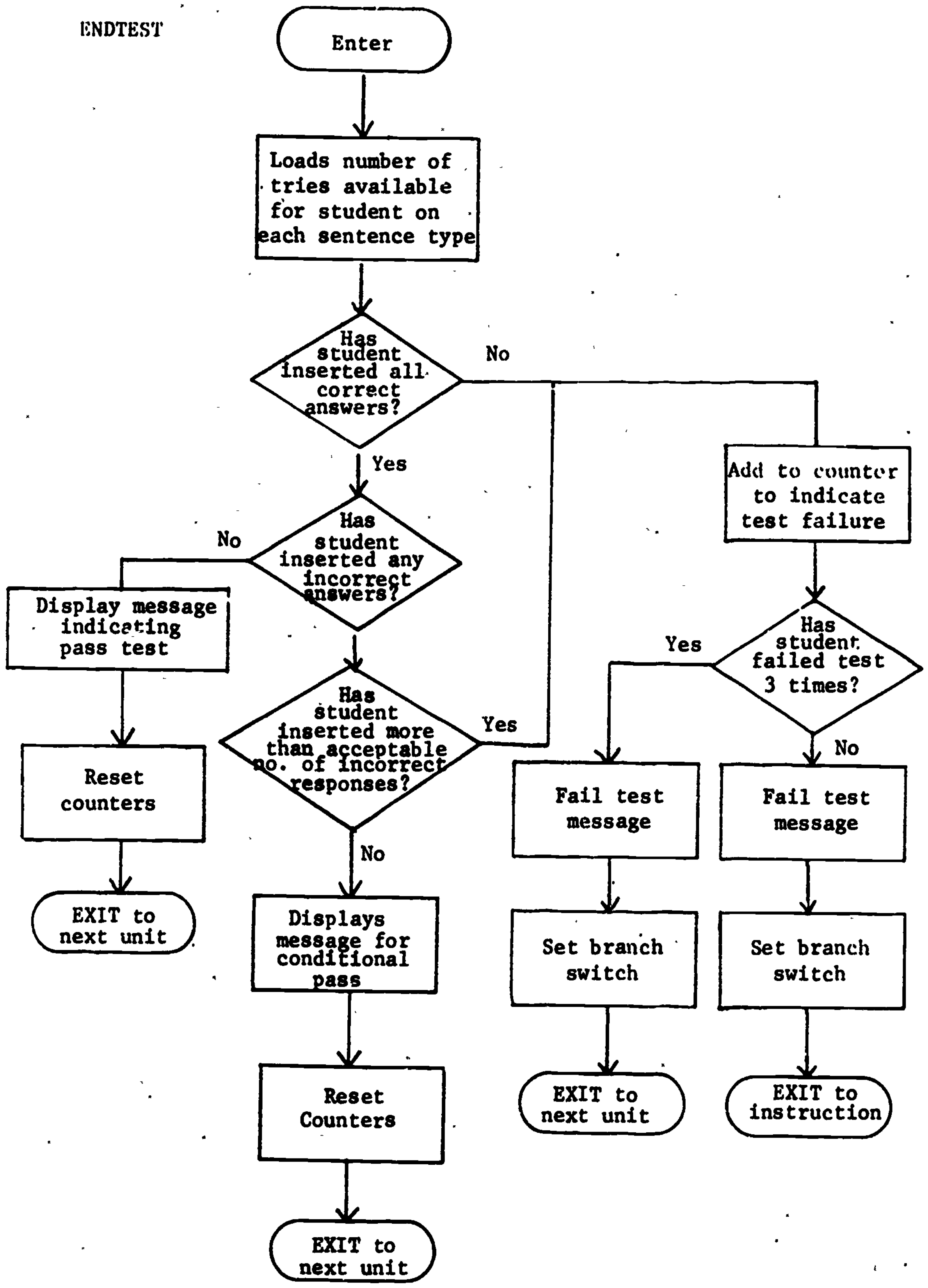
This function is called to give a sentence of a specified prototype to the student for punctuation additions. DØEND is set previously; DØEND determines how many sentences of this type are presented to the student in drill until he gets one correct. On tests DØEND is set at one. For the present, DØEND is set at five during the instructional drill.

NEWTEST


Enter
Has student failed this test?
Yes
Sets switch to indicate posttest
No
Sets switch to indicate pretest
Is student on pretest?
No
Yes
Student is given review option before posttest
Resets counters for test
Does student want review?
Yes
No
EXIT
Displays introductory message for test
Sets chances at sentence type to 1 (was 5 in course)
EXIT

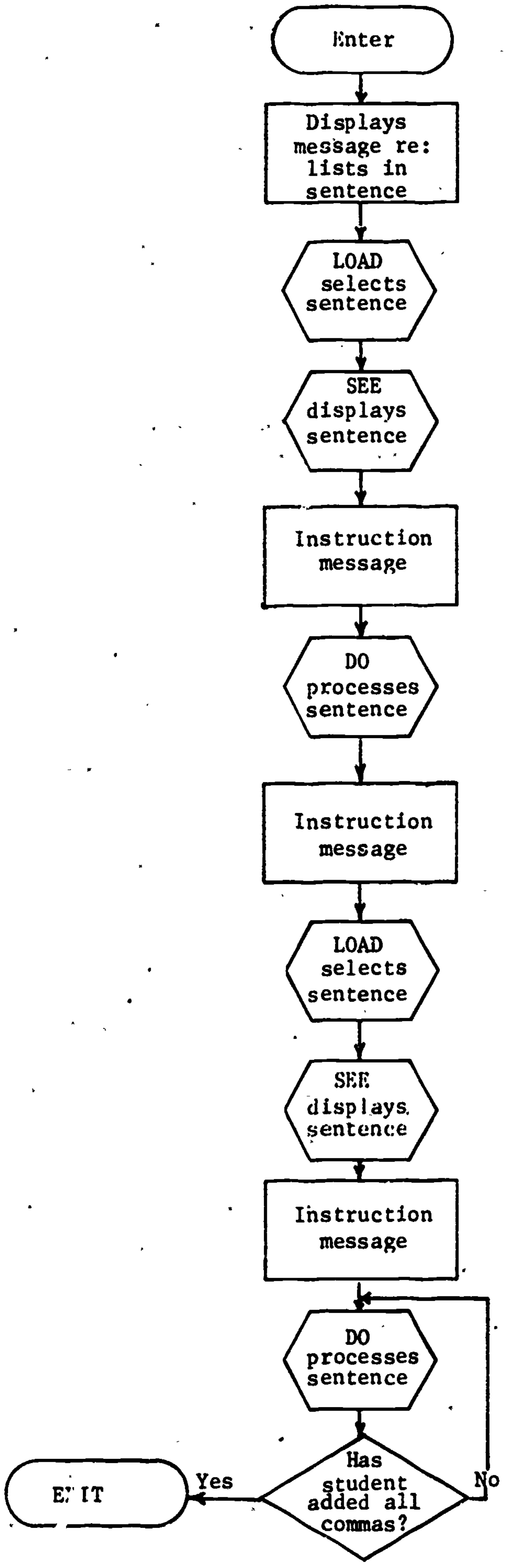
This function is called prior to any test. It displays messages and gives the student an option to review before the posttest. It also clears counters and sets the loop used in instruction (DØEND) to one for testing purposes. No parameters are specified.

ENDTEST


Enter
Loads number of tries available for student on each sentence type
Has student inserted all correct answers?
No
Yes
Has student inserted any incorrect answers?
No
Display message indicating pass test
Reset counters
EXIT to next unit
Has student inserted more than acceptable no. of incorrect responses?
Yes
No
Displays message for conditional pass
Reset Counters
EXIT to next unit
Add to counter to indicate test failure
Has student failed test 3 times?
Yes
No
Fail test message
Set branch switch
EXIT to next unit
Fail test message
Set branch switch
EXIT to instruction

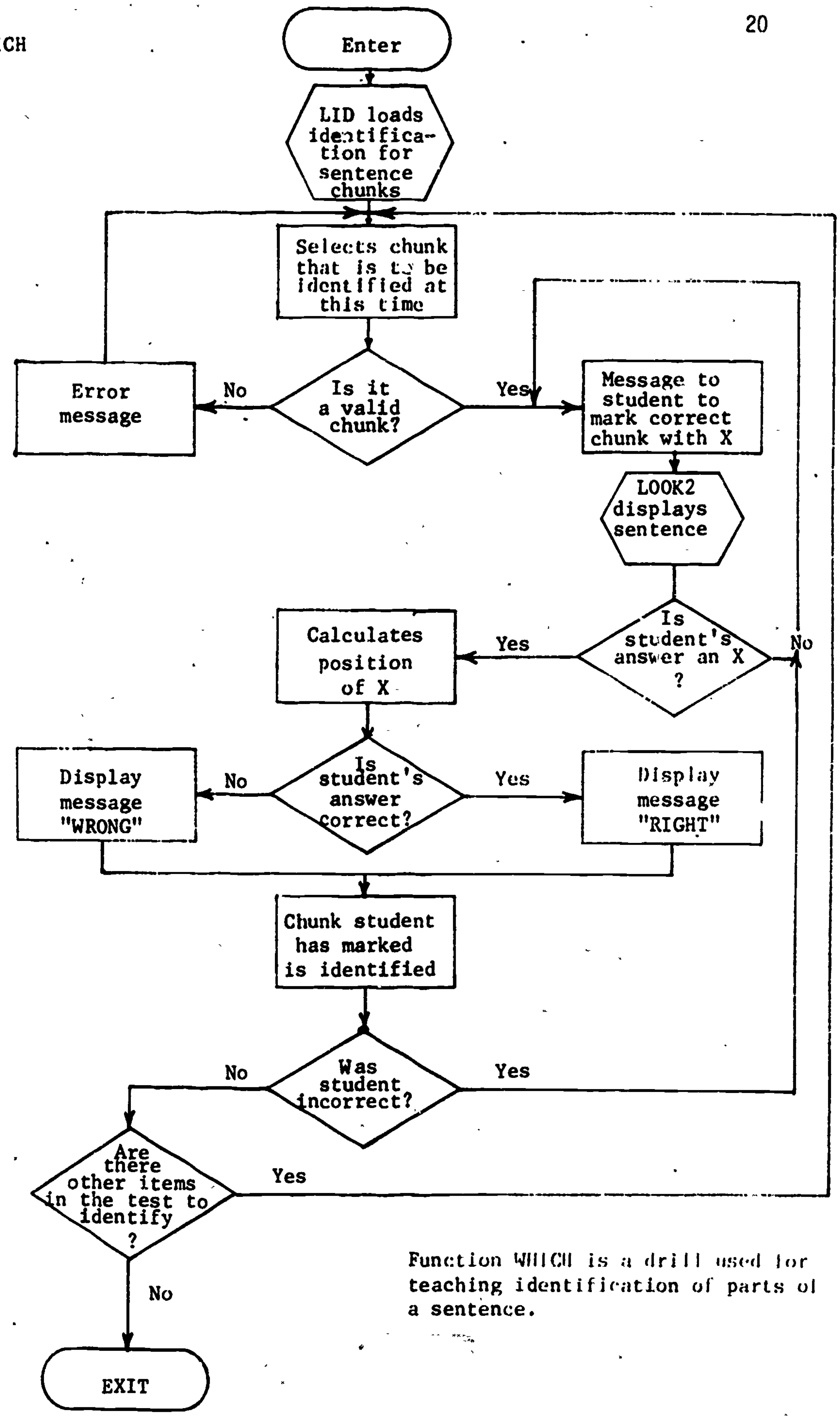
This function is called at the end of a test. It checks criterion score and controls branching, messages, and progression.

LIST2



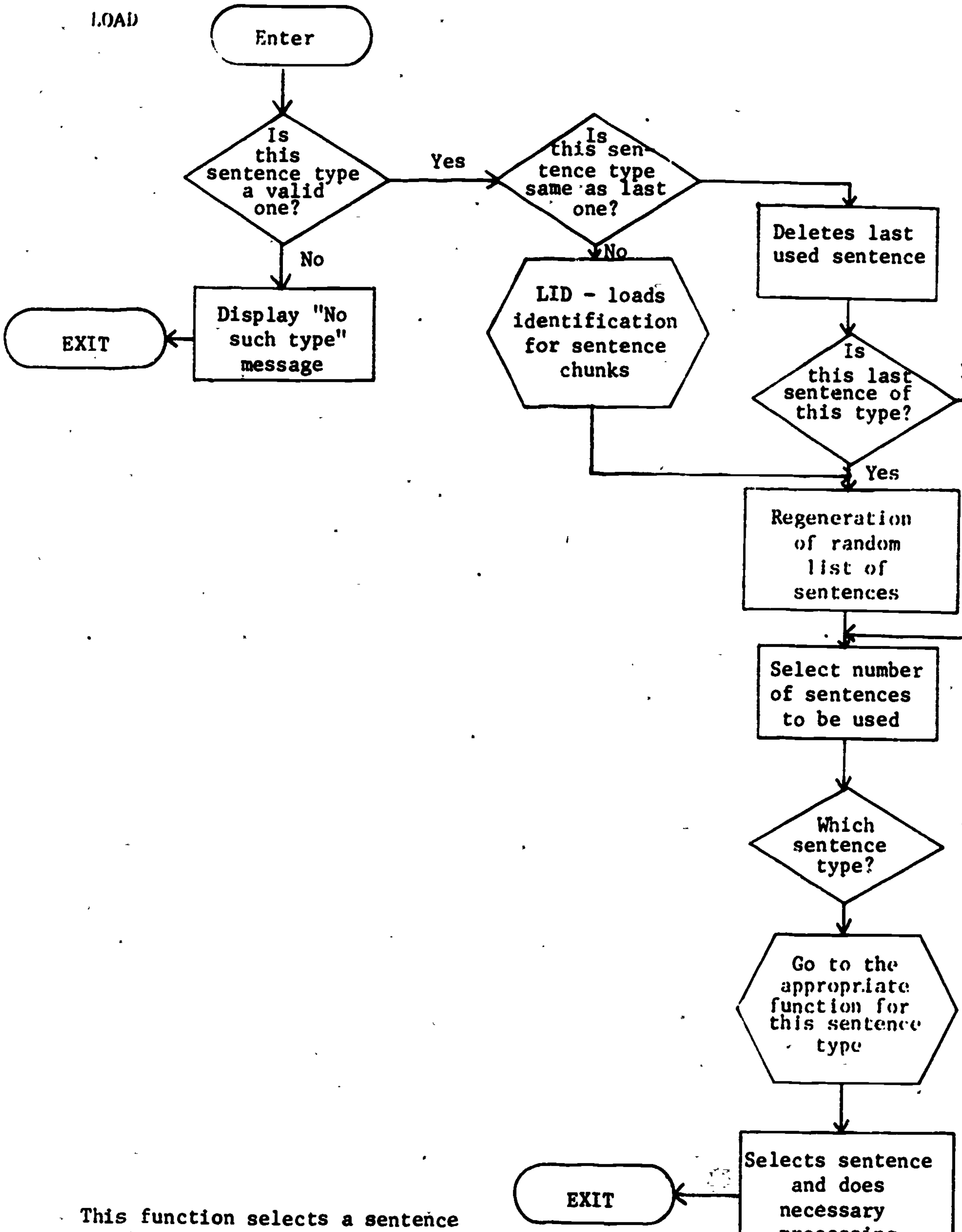
LIST2 prepares and displays a sentence for correction by the student. Instructional messages are embedded. (LIST is similar in form, but instruction is different.)

WHICH

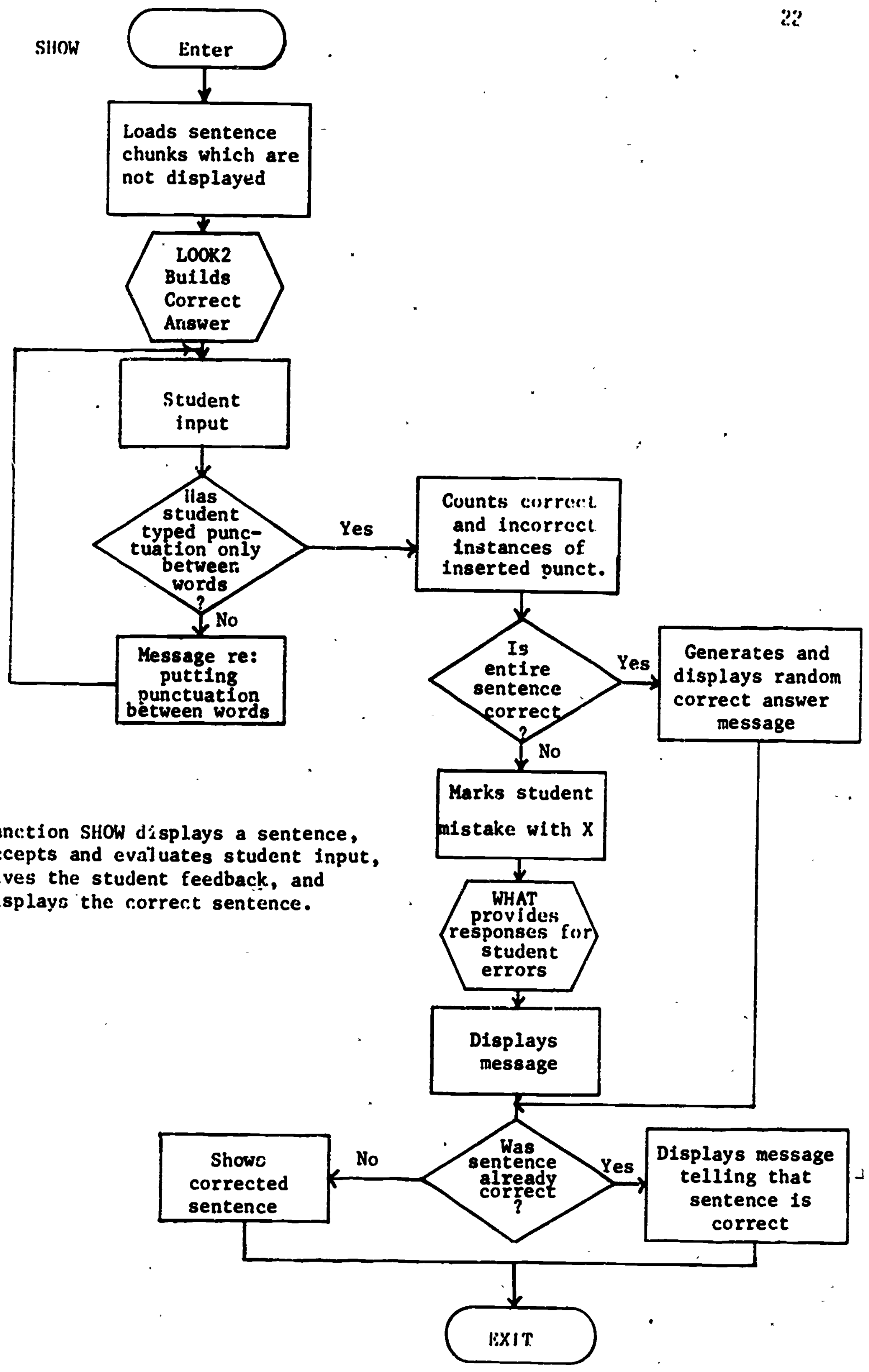

Enter
LID loads identification for sentence chunks
Selects chunk that is to be identified at this time
Is it a valid chunk?
No
Error message
Yes
Message to student to mark correct chunk with X
LOOK2 displays sentence
Is student's answer an X ?
Yes
No
Calculates position of X
Is student's answer correct?
No
Display message "WRONG"
Yes
Display message "RIGHT"
Chunk student has marked is identified
Was student incorrect?
No
Yes
Are there other items in the test to identify ?
Yes
No
EXIT

Function WHICH is a drill used for teaching identification of parts of a sentence.

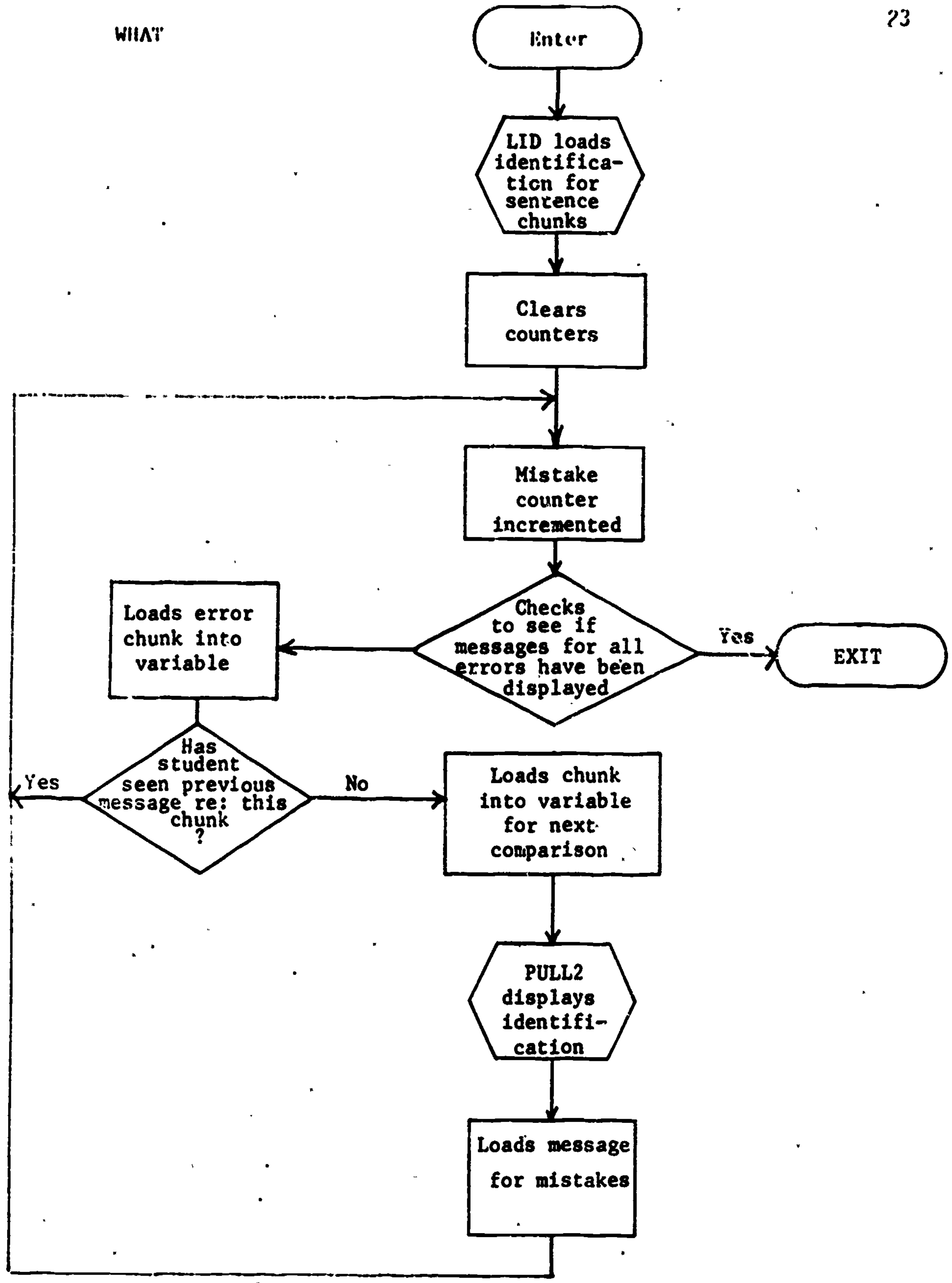
LOAD


Enter
Is this sentence type a valid one?
Yes
No
Display "No such type" message
EXIT
Is this sentence type same as last one?
No
LID - loads identification for sentence chunks
Deletes last used sentence
Is this last sentence of this type?
No
Yes
Regeneration of random list of sentences
Select number of sentences to be used
Which sentence type?
Go to the appropriate function for this sentence type
Selects sentence and does necessary processing
EXIT

This function selects a sentence of the appropriate prototype and processes the sentence before it is used.

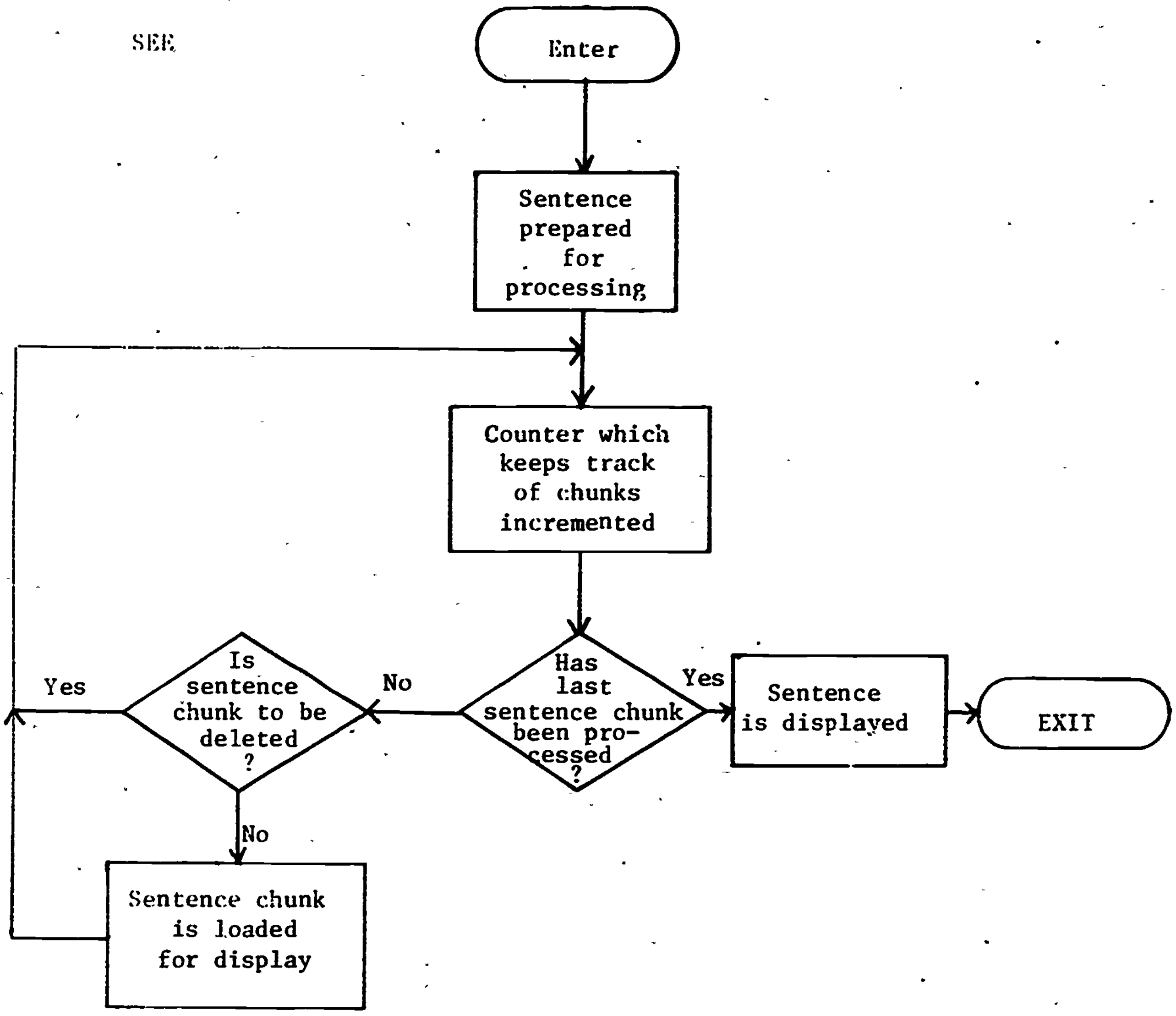

SHOW
Enter
Loads sentence chunks which are not displayed
LOOK2 Builds Correct Answer
Student input
Has student typed punctuation only between words ?
Yes
No
Message re: putting punctuation between words
Counts correct and incorrect instances of inserted punct.
Is entire sentence correct ?
Yes
No
Generates and displays random correct answer message
Marks student mistake with X
WHAT provides responses for student errors
Displays message
Was sentence already correct ?
No
Yes
Shows corrected sentence
Displays message telling that sentence is correct
EXIT

Function SHOW displays a sentence, accepts and evaluates student input, gives the student feedback, and displays the correct sentence.


Enter
LID loads identification for sentence chunks
Clears counters
Mistake counter incremented
Checks to see if messages for all errors have been displayed
Yes
EXIT
Loads error chunk into variable
Has student seen previous message re: this chunk ?
Yes
No
Loads chunk into variable for next comparison
PULL2 displays identification
Loads message for mistakes

This function provides responses for student errors.

SEE



Enter
Sentence prepared for processing
Counter which keeps track of chunks incremented
Has last sentence chunk been processed ?
Yes
Sentence is displayed
EXIT
No
Is sentence chunk to be deleted ?
Yes
No
Sentence chunk is loaded for display

Function SEE prepares and displays sentence with appropriate sentence chunks omitted.