DOCUMENT RESUME

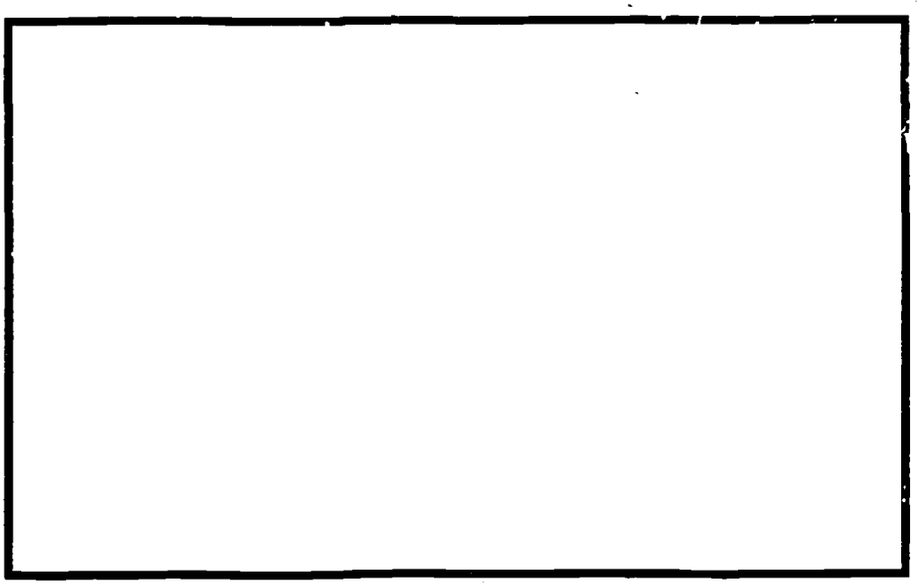
ED 084 880 EM 011 704

AUTHOR Freed, Michele M.
TITLE Generation of Punctuation and Usage Exercises in Freshman English Using a Sentence Pool (PUNCT2-CW). Technical Report Number Six.
INSTITUTION Texas Univ., Austin. Computer-Assisted Instruction Lab.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO TR-6
PUB DATE Dec 70
NOTE 15p.

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Computer Assisted Instruction; *Computer Programs; *Data Bases; English Instruction; Pattern Drills (Language); Programing; *Punctuation
IDENTIFIERS *PUNCT2CW; Sentence Pools

ABSTRACT

The purpose of PUNCT2-CW is to test the feasibility and practicality of a data base system of computer-assisted instruction in English. To save time and to provide comparison, this course was based on the objectives, format and logic of an earlier course, PUNCT. The present data base is a sentence pool composed of certain sentence patterns called prototypes, each with a unique call number. A program author can call a type of sentence by using the prototype name. By employing certain macros, he can use the sentences from the pool as if they had been hard coded. Original sentences can be quickly and simply added to the pool. Flow charts document the process from the time a sentence is called from the pool until it has been displayed and finally answer processed. The description of how the macros are used, and what must be included in a macro call, is a more detailed account expanding drills and providing new sentences within the already coded data base, but a modified system of macros is needed to simplify recall from the sentence pool. The speed of the system will be improved by translation into the APL programing language. (Author/SL)



THE UNIVERSITY OF TEXAS AT AUSTIN

Computer Assisted Instruction Laboratory

AUSTIN

GENERATION OF PUNCTUATION AND USAGE EXERCISES IN FRESHMAN ENGLISH USING A SENTENCE POOL (PUNCT2-CW)

TECHNICAL REPORT NO. 6

Michele M. Freed

December 1970

Supported By:

THE NATIONAL SCIENCE FOUNDATION
Grant GJ 509 X

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

GENERATION OF PUNCTUATION AND USAGE EXERCISES IN FRESHMAN ENGLISH USING A SENTENCE POOL (PUNCT2-CW).

The purpose of PUNCT2-CW, a course based on an earlier course PUNCT, is to test the feasibility and practicality of a data base system of computer-assisted instruction in English. This document discusses the existing data base and its implementation, as well as future adaptations and uses of such a system.

The present data base is a *sentence pool* composed of certain sentence patterns called *prototypes*. A program author can call a type of sentence by calling the prototype name. By using certain macros, he can use the sentences from the pool as if they had been hard coded. Associated with each sentence pattern is a *profile* of characteristics, which is unique for each type of sentence and is given a unique two-character alphabetic name, called a *prototype code*. The sentences of each type are classified under the appropriate code. Each sentence has a sentence identification number, and each must contain the complete series of *units* which make up that particular prototype. In the present sentence pool a unit consists of a group of words, a single word, or a punctuation mark.

The sentence pool is composed of a number of prototypes, now limited to those called by PUNCT2-CW. Each prototype has distinct characteristics and is given a unique name. A mutation of the main prototype is called a *derived prototype* and differs from its main prototype in having one or more of the main prototype units deleted. For example, the main prototype AA contains the following units: a head string, day of week, comma, date of month, comma, year, comma, and tail string. A derived prototype of this might be called AAA and contain: a head string, date of

month, comma, year, comma, and tail string. The author might find the derived prototype useful for illustrating certain points. If he plants to use a main prototype making the same deletions each time, the author should establish a derived prototype for his convenience.

An author who wishes to use the present data base system is given a list of prototypes, a description of how to use the macros, and an explanation of how to add sentences to the sentence pool. He can write his instruction independently and then use the prototypes to call in the appropriate sentences for examples, drills, and quizzes. The author learns how to call, manipulate, and display the sentences. Since he can answer process sentences without hard coding each one, his drill capabilities are greatly expanded, and coding becomes more efficient. He can quickly and simply add sentences (which suit his needs) to the sentence pool. A secretary or clerk can use subroutine *zipzip* to prepare sentences for the pool.

Figures 1, 2, and 3 document the process which takes place from the time the sentence is called from the sentence pool until the sentence has been displayed and finally answer processed. The following description of how the macros are used and what an author must include in the macro call is a more detailed account than the overview provided by the flow charts. Macro *qa* is used for the initial sentence call, and the following statement is an example:

cm *qa* ", AA ", 0 ", &inq ≠

The action code for macro *qa* can be either 1 or 3; however, 3 is used to recall a sentence. The author normally calls a sentence by its prototype code, in this instance AA. If the author were calling a specific sentence,

he would replace the prototype code with the sentence identification number. This substitution can be especially useful if the author wishes (a) to call a sentenced (loaded into the pool) that does not fit a prototype, (b) to use one particular sentence, or (c) to call the same sentence at a number of different points within the program. The zero following the prototype code indicates that a derived form of the prototype is not being called. If it were being called, the code numbers would indicate which units of the main prototype were deleted to form the derived prototype. The next parameter of the macro contains the *action indices*, which specify the number of the sentence units that are to be acted upon; in this case, because of action code 1, the units are to be deleted. In the currently coded exercises, the action indices usually refer to sentence units which are commas. If the code had indicated the use of a derived sentence, the sentence pulled from the pool would have some predesignated missing units. However, the derived sentences are complete and correctly punctuated, and deletions would still be made through the use of action indices if the sentences were to be used for exercises concerning the addition of punctuation. The final macro parameter contains a *return label* to which the logic returns after the sentence selector logic and the sentence processing logic have been executed. From macro *qa*, the program branches to the label of the prototype name in the sentence selector logic specified by the macro. The sentence selector *prefix* is written into buffer 5, and the position of the *suffix* stored in buffer 2 is made available. The program then checks c30 for a derived sentence code. If the call is for a derived sentence, new values are written into buffer 5 (the prototype prefix) and c20 (the position of the suffix in buffer 2). The suffix in buffer 2 is placed in

c28. If the action code is 1, the suffix is reset for a new sentence, and, if the action code is 2 or 3, the suffix is set for the last sentence of this prototype called. In other words, action codes 2 and 3 are used to recall sentences. The sentence suffixes come from the pool in descending order. If all sentences of a prototype are used, the suffix counter is reinitialized, and the program will start through the sentences of that prototype again. Macro *pi* reinitializes the prototype suffix. The new suffix--new because it was reset for the new sentence or reinitialized by macro *pi*--is returned to buffer 2. The sentence selector suffix is attached to the prefix in buffer 5, and the entire label is stored in *return register* 5. The program branches to this label in the prototype-sentence correspondence table. In the table each possible generated label is associated with a sentence identification number, and each sentence identification number is associated with a particular sentence. Subroutine *zipzip* prepares the sentences for entry into the pool through macro *sp*. *Zipzip* is executed in author mode at a typewriter terminal. The operator enters the sentence identification number and then the sentence units one at a time at the computer's command. A *blank unit* (containing only an *eob*) indicates that all sentence units have been entered. The first output of *zipzip* is the sentence identification number, after which a numeric profile of the sentence is typed. The first digit of this profile indicates the number of units in the sentence, and each of the following numbers describes one sentence unit. For example, the numeric profile 3 4106 1010 903 indicates that the sentence has three units (profile count): (a) The first unit contains 41 characters, and its last word contains 6. (b) The second unit has 10 characters, and the last word contains 10. (c) The last unit has 9

characters, and the last word has 3. *Zipzip* also types the sentence with units indicated by unit markers (in this case, an *X*). After the sentences are loaded in this form, the program branches to the sentence processing logic, where the sentences are prepared for answer processing and display. If the action code is 1 or 3, a *binary string* is placed in buffer 2 in front of the action indices which were loaded by macro *qa*, and buffer 4 is cleared. The action indices are used to compute *positions* in the binary string. For every action index in a *qa* macro call, a unit is deleted from the sentence. If an action index is 3, then the third 1 in the binary string is replaced by a zero and so on for each action index. The profile count is then used to compute the end of the binary string. The binary string is chopped off and collapsed to the size of the sentence. From this binary string, the units for deletion (the units marked by zeros) are identified. The profile counts in buffer 0 for all zeros in the binary string are converted to zero. If the action code is 1, the sentences enter a subroutine which uses the binary string and the profile count to establish response identifiers (*response 3-tuples*) in buffer 4. A 3-tuple is created for the word previous to a deleted unit. This is done because the student, using a light pen, must touch a word that he wants the punctuation mark to follow. The first number of the 3-tuple is the relative row of the word (relative to the beginning row of the sentence); the second is the number of the columns in the response area; and the last is the beginning column of the response area. These response identifiers permit the definition of a light pen response area through counters. Each sentence, whether called by macro *qa* or *qb*, goes through subroutine ClEaN. The selected sentence is edited, and a character count is taken. Carriage returns and indexes are inserted

so that the display of the newly edited sentences will fit on the screen. At this time, the return label is in return register 2, buffer 2 contains the sentence selector label suffixes, buffer 3 contains a binary string which reflects the sentence unit omissions, buffer 4 holds the response 3-tuples, and buffer 5 contains the processed sentence which is ready for display. Control is returned to the calling macro by the return register 5 label. The author then uses counters 3 (row) and 24 (number of rows) to determine where the sentence is to be displayed and uses function *dt* to display it. The student is now able to respond by touching a word which should be followed by a comma. Function *pen* places the coordinates of the light pen response into counters 1 and 2; macro *qp* examines the contents of these counters and determines if the student's response matches any of the response area identifiers for the sentence. If no match occurs, the student is given a standard *un* message and the opportunity to respond again or to proceed to the next sentence. If a match does occur, a counter is set by macro *qp* to indicate which 3-tuple was matched. By using this counter, the author branches to the macro (*qb*) that will restore the appropriate comma.

The macro call to *qb* resembles the call to *qa*:

cm *qb* " / 2 ", AA ", 0 " , 3 ", 8inq4 ≠

The action code 2 shows that a unit will be restored to the last called sentence of prototype AA. The zero after the prototype name indicates that the sentence is not a derived form of AA. The action index indicates that the third unit of the sentence is to be restored and that control will be returned to the macro *qb* by a branch to label 8inq4. This macro works similar to macro *qa* except that it manipulates the previously called sentence

of the prototype instead of a new one. The sentence selector prefix is placed in buffer 5, and the position of the sentence selector suffix is made available. If *qb* calls for a derived sentence, new values are written into buffer 5 and counter 20. The sentence selector label suffix is extracted from buffer 2, and with action code 2 (the same is true for action code 3 of macro *qa*) the suffix value is that of the last sentence called of this prototype. The suffix is attached to the sentence selector prefix in buffer 5, and the entire label is stored in return register 5. The program branches just to the label in return register 5 and then from this label in the sentence pool correspondence table to the sentence identification number. The sentence is loaded by macro *sp* as described above. With the sentence in buffers, the branching proceeds to the sentence processing logic. Since this sentence is being recalled, there is no need to create a new binary string. The existing binary string is used to identify the unit named by the action code for restoration. After the unit has been restored, the sentence once again goes through subroutine ClEaN to be prepared for display. Control is then passed to the return label specified by the calling macro. The newly processed sentence is displayed, and the student is then able to respond again. The author may wish to recall a sentence using macro *qa* with the action code 3. The process described above is followed, but the author is able to restore more than one unit at a time, delete additional units, or merely recall the sentence in its current form. If he wishes to display the complete sentence, he substitutes an *n* for the action indices in the macro call. When control is returned to the calling macro, a statement is needed to display the sentence.

Two units of PUNCT were adapted to the data base system. The instructional sequence and prose remained the same; however, the hard coded sentences were replaced by calls to the sentence pool. This alteration presents new examples and new exercises to the PUNCT2-CW student and offers a new posttest each time he reviews the instruction.

PUNCT was chosen for a number of reasons: (1) The emphasis of the project was on developing a data base system to be used for other courses in English, rather than on writing a single new English course. Since the course objectives, logic, and formats were already established, considerable time was saved, although some changes had to be made (2) By using a prepared course the authors were able to design a limited data base to determine if the general concepts were feasible before they continued to investigate the system's adaptability. PUNCT provided such a base. A generalized system that could process the addition of punctuation to sentences drawn from a sentence pool provided a base that would give experience in coping with unanticipated problems and in foreseeing problems that might arise in extensions and new applications of this generalized system. (3) Since the text and logic were previously coded and debugged, using PUNCT proved to be time saving. Had the data base been written without using a course as its base, it would have been more difficult to limit. This first attempt revealed the changes and extensions necessary to make the system more comprehensive. (4) The adaptation of segments of an already prepared course provided a basis for comparison. A study could be conducted to determine whether or not new examples helped the student when he reviewed the course. With some modifications, the PUNCT2-CW student could request

more drill. A study could also determine if students wanted more drill than was provided in PUNCT and whether or not this additional drill would affect the student's performance. Once the data base system became operational, adding and calling new sentences required less time (of the author, coder, and keypunch operator) than the hard coding and debugging of PUNCT sentences. The modular-designed course can easily be debugged, while PUNCT is almost impossible to debug completely.

A generalized system has many advantages. For example, the instructional logic and the data base are written, expanded, and modified independently. In addition, an instructor can compose new prose but use an already coded data base. He also has the alternative to use the instructional sequence (as was done in PUNCT2-CW) and prepare a new data base. If a teacher wishes to use sentences and examples that are more relevant to his class, he can easily change the pool. He can expand drills without putting in new coding other than that which branches the student into a unit which calls sentences from the pool.

Some disadvantages when using PUNCT were also discovered. Since PUNCT was not written with a generalized system as its base, the course was very specific in some instances. Detailed messages for wrong answers and sometimes even for correct answers had to be replaced by messages appropriate to the prototype rather than to the specific sentence. The drill expansion capabilities of the system were not used since each sentence was hard coded. Some instruction examined particular sentences in such detail that they could not be pulled from the pool to serve as examples. PUNCT could not utilize all of the advantages of a data base system because the course was not written with the generalized system in mind. This system

now operates with a string in which units can be deleted and restored. This system could be improved by certain modifications. The most obvious problem with PUNCT2-CW is that using sentences from the pool is still complicated. Macros are used to call and correct the sentence, but the author must decide where the sentence will be displayed. Branching logic to correct the sentence if it is one of many can become complex and confusing. A modified system of macros should be written to simplify the use of the sentence pool. Also, the current method of identifying *wa* responses is difficult and inefficient. A new variable added to the macros would permit referencing a sentence element without deleting it. The option of replacing one element with another would be useful in other exercises of this type. Since this program is very slow on the IBM 1500 Coursewriter System, it is now being translated into the APL programming language. A faster and more efficient version of PUNCT2-CW will be available in APL.

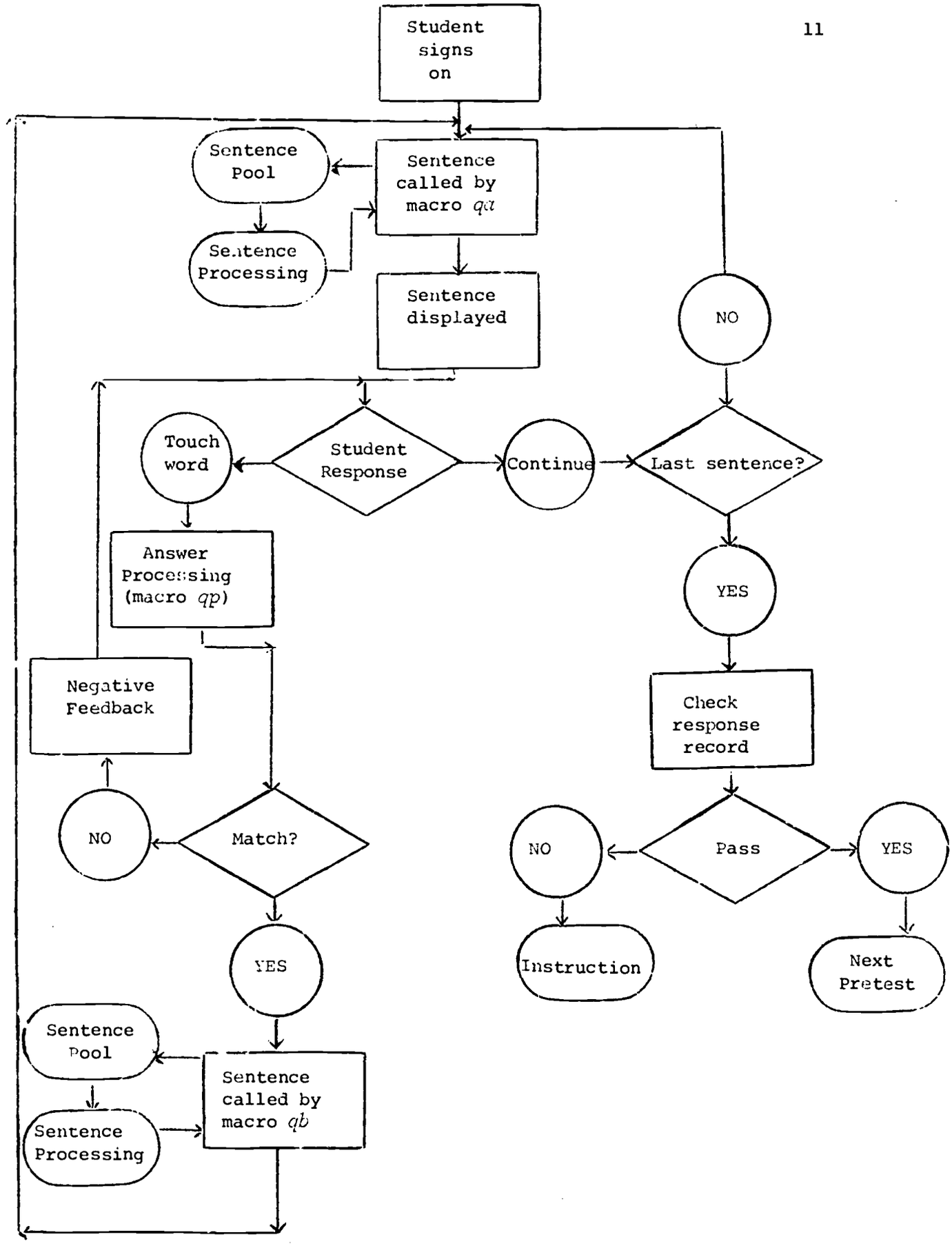
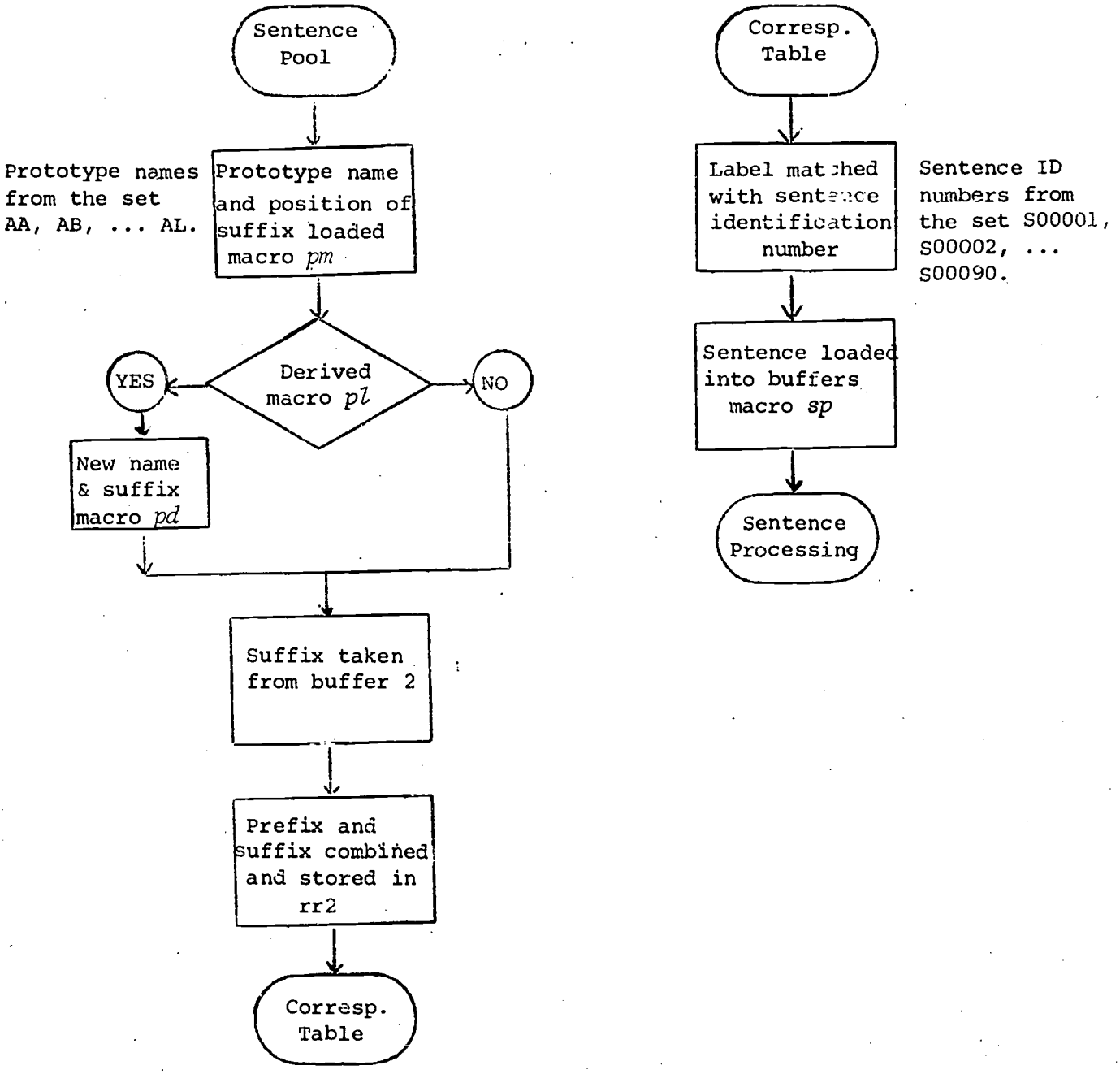

Student signs on
Sentence Pool
Sentence called by macro qa
Sentence Processing
Sentence displayed
NO
Touch word
Student Response
Continue
Last sentence?
Answer Processing (macro qp)
YES
Negative Feedback
Check response record
NO
Match?
NO
Pass
YES
YES
Instruction
Next Pretest
Sentence Pool
Sentence called by macro qb
Sentence Processing

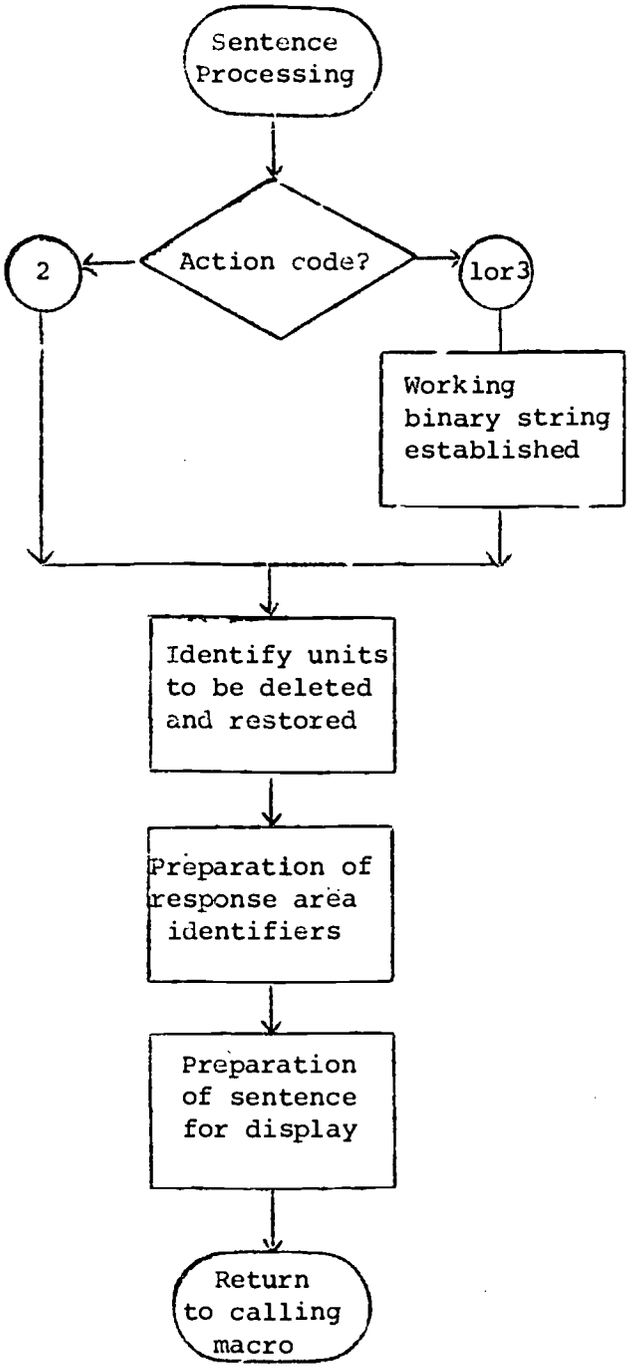
FIGURE 1



The macro call branches to the sentence selector logic which gives the program access to a sentence which satisfied the prototype requirements presented to it. Each sentence selector label is associated with a specific sentence identification number. This six-character sentence identification prototype-sentence correspondence table sets up a one to one correspondence between the generated prototype label and the sentence identification number.

Control is transferred to the label of the sentence identification number. The logic of macro *sp* is executed and buffer 3 is loaded with the sentence profile and buffer 5 is loaded with the desired sentence in its unprocessed form.

F I G U R E 2



After the sentence has been loaded, control is transferred to the sentence processing logic. At this point, a binary string is generated in buffer 3 from the action indices in buffer 3 (loaded by macro *qa*). The sentence profile is marked against the binary string and response pointers are computed and placed in buffer 4. The raw sentence in buffer 5 is marked against the binary string and is replaced by an intermediate version of the desired sentence. The sentence in its intermediate form is processed for future display from buffers. Control is returned to the original calling macro with buffer 5 containing the desired sentence and buffer 4 containing the response pointers.

F I G U R E 3