ESD-TR-68-143, VOL III
ESTI FILE COPY

ESD RECORD COPY
RETURN TO
SCIENTIFIC & TECHNICAL INFORMATION DIVISION
(ESTI), BUILDING 1211

ESD-TR-68-143, Vol. III

EVOLUTIONARY SYSTEM FOR DATA PROCESSING

THE CAINT EXECUTIVE LANGUAGE AND INSTRUCTION GENERATOR

Charles T. Meadow
Douglas W. Waugh
Gerald F. Conklin
Forrest E. Miller

ESD ACCESSION LIST
ESTI Call No. AL 61084
Copy No. 1 of 1 cys.

AL 61084

January 1968

COMMAND SYSTEMS DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

This document has been approved for public release and sale; its distribution is unlimited.

(Prepared under Contract No. F19628-67-C-0254 by Center for Exploratory Studies, International Business Machines Corporation, Rockville, Maryland.)

AD670840

LEGAL NOTICE

When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

FSC 68-0670/

ESD-TR-68-143, Vol. III

EVOLUTIONARY SYSTEM FOR DATA PROCESSING

THE CAINT EXECUTIVE LANGUAGE AND INSTRUCTION GENERATOR

Charles T. Meadow
Douglas W. Waugh
Gerald F. Canklin
Farrest E. Miller

January 1968

COMMAND SYSTEMS DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscam Field, Bedford, Massachusetts

This document has been approved for public release and sale; its distribution is unlimited.

(Prepared under Contract No. F19628-67-C-0254 by Center for Explaratory Studies, International Business Machines Corporatian, Rockville, Maryland.)

FOREWORD

This report presents the results of a study of the specifications for an information system intended to support the design, production and maintenance of large computer programming systems. Called Evolutionary System for Data Processing, or ESDP, it was begun as an internal IBM project in 1965 by the Center for Exploratory Studies of the Federal Systems Division and continued under Air Force sponsorship during 1967 and early 1968.

This work has been performed under contract number F19628-67-C0254 for the Electronic Systems Division, U.S. Air Force Systems Command. The project monitor was Mr. John Goodenough, ESLFE.

The authors wish to express their appreciation for the encouragement and assistance provided by Dr. John Egan, formerly of ESD, and their colleagues Dr. Harlan D. Mills and Mr. Michael Dyer.

This report is in four volumes: Volume 1, System Description; Volume 2, Control and Use of the System; Volume 3, The CAINT Executive Language and Instruction Generator; and Volume 4, Programming Specifications. This report was submitted on January 31, 1968.

This report has been reviewed and is approved.

Sylvia R. Mayer

SYLVIA R. MAYER
Project Officer

William F. Heisler

WILLIAM F. HEISLER, Col, USAF
Chief, Command Systems Division

ABSTRACT

ESDP is a proposed system whose purpose is to acquire, store, retrieve, publish and disseminate all documentation, exclusive of graphics, concerned with a large computer programming activity. Documentation is deemed to consist, not only of final or formally published after-the-fact reports, but of working files, design and change notices, informal drafts, management reports--in fact, the entire recordable rationale underlying a programming system. Maximum attention has been concentrated on the means of acquiring and organizing documentation. Two major, complementary approaches are proposed. The first is called Program Analysis and is a process of extracting documentation directly from completed programs. The second is called Computer Assisted Interrogation and is a process of eliciting information directly from the programmer, through on-line communication terminals. The former provides canonical data about the program's structure. The latter provides explanatory material about all aspects of the program, and in the absence of canonical data, may provide tentative structural information as well. The conclusion of the study group is that ESDP is a feasible concept with present-day technology and that it will materially benefit using organizations in the production of programs and in guiding their evolution as requirements change. Its value will be greater for larger organizations, whose internal communications difficulties tend to cause truly gigantic inefficiencies. Its implementation as a support system for such projects would require a significant quantum of investment in order to produce these benefits and is predicated on the use of a computer system dedicated solely to the use of ESDP.

Volume 3

The CAINT Executive Language and Instruction Generator

		Page
	I	
	THE CAINT EXECUTIVE LANGUAGE	1
1.	Introduction	1
2.	Language Requirements	1
3.	Language Elements	3
4.	Response Processing	7
5.	Review and Update	9
6.	Special Responses	10
7.	Documenting CEL Programs	11
	II	
	TECHNIQUES OF INTERROGATION	14
1.	The Data Structure	14
2.	Interrogation Sequence	15
3.	Considering the Responder	15
	III	
	INSTRUCTION IN ESDP	20
1.	Objective	20
2.	An Instruction Generating Program	21

IV

INSTRUCTIONAL PROGRAMS 27

1. The Program Model 27

2. An Example 28

3. Composition of a Question 31

4. Composition of the Program 36

5. Using the System 36

V

BIBLIOGRAPHY 37

ILLUSTRATIONS

Figure		Page
1	A Sample Interrogation Program	5
2	Response Processing	8
3	Example Of CEL Program Documentation	12
4	A Question in CAI Form	26
5	A Question in PI Form	26
6	A Unit of Instruction or Question	29

I

THE CAINT EXECUTIVE LANGUAGE

1. Introduction. In this volume we describe the conversational programming language, called the CAINT Executive Language or CEL, and describe its application to interrogation and instruction.

The reason for developing CEL was to provide a facility for programmers that would enable them to write conversational programs easily. A conversational program in the context used here is a computer program which carries on a conversation between man and machine for the purpose of exchanging information, and for which there is a need for generality in expressing the conversational elements. Generally, in conversational programming systems, interaction is between a programmer or designer one hand and an operating system on the other, and conversations are limited to computer programming, graphic design, text editing, or whatever application has been built in at the operating system level. In computer assisted instruction, to which CAINT is similar, conversations cover a wider range of topics and there is an intermediary program, the course, written by someone other than the programmer of the operating system. The student taking a CAI course, for all practical purposes, is aware only of the course and the particular limitations and capabilities it has. He is not necessarily aware of the operating system.

In CAINT, as in CAI, the student, or responder or documentor, deals not with the operating system but with a program, or "course," written by an executive programmer. Conversational techniques are the result of the style of the executive programmer, not the basic system logic.

Sections I and II of this volume are primarily devoted to a description of the language and to techniques for its use in interrogation. While its use in instruction imposes no additional requirements or constraints, we have devoted most of our effort on instruction to the generation of instruction programs. Sections III and IV describe our approach to instruction and the Instruction Generator, a CEL program that produces CAI or PI courses.

2. Language Requirements. A conversational programming language, in this context, does nothing different from an ordinary programming language. The difference is in emphasis and in the relative ease of writing certain kinds of program expressions or in accomplishing certain kinds of program steps. The CAINT Executive Language is a general language and could be used for almost any application, albeit inefficiently in most cases. Similarly, almost any other language could perform the logical functions of interrogation or instruction but, we think, seldom with the ease and efficiency of this PL/I dialect.

Let us consider briefly what the major conversational functions are and then review each in greater detail. The conversational programmer would like:

a. To be able to decide what question the machine should ask of the man at any given point in the conversation and to be able to do this on the basis of any definable function of the program's data base, including, but not limited to, the response to the last question.

b. Not to have to store the exact text of every question before asking it. Hence, the program should be able to assemble a question either from fragments or skeleton questions or from the data base.

c. To make actual presentation of the question through a suitable output device, elicitation of an answer, and storage of the answer as mechanically simple as possible because these are such high frequency operations.

d. To permit a wide range of response types such as: multiple choice, a single number, full text.

e. To have a simplified method for programming of response analysis. The author should be able to decompose an answer into an array of words, test a response against an array of possible values, or perform any of a large number of other text-handling functions with a minimum of programming effort.

f. To have a simplified logic for coping with unrecognizable (unanticipated by the author) answers. This would permit the writing of extensive programs for keeping track of unrecognizable answers and varying the content of machine initiated messages depending on the number of consecutive unrecognizable responses.

g. To have a large number of functions on which branching decision can be made. Branching within the conversational program is, of course, a requirement. Branching decisions should be able to be made on the basis of any definable function of the data base.

h. To have any portion of the conversational program able to be operated as a subroutine so that it may be executed in other than its normal sequence. This permits reviews and updating.

i. To have the student be capable of controlling program execution, to some extent, by the use of non-responsive answers to questions. A non-responsive answer is one which substitutes a command for the response called for. Such commands include: GO TO, SIGN OFF, QUERY and BACK. The last of these enables a responder to reverse the sequence of operation of the program and step backward through the last n questions. It can be used to recover from the situation where the responder gives a

legal but incorrect response, then discovers that he did not mean to give that answer and now wishes to pursue the logical path which would have resulted had he given the correct answer the first time.

3. Language Elements. The basic language used for CEL is PL/I. CEL is a subset of PL/I with several specialized subroutines, also written in PL/I, and several usage conventions that simplify programmer bookkeeping and review or updating operations. CEL operates under Operating System/360 and can achieve multi-terminal operation through the facilities of OS.

In experimental work to date, we have used the IBM 2260 cathode ray tube display terminal as the man-machine interface. Hard copy terminals, such as the IBM 1050, can be used by linking the PL/I programs with the Queued Telecommunications Access Method (QTAM) of OS. This linkage will have to be accomplished through an assembly language program. It has not been accomplished to date, but there are no theoretical problems anticipated in doing so. The desirability of having a hard copy terminal in addition to a CRT is, we think, established. In spite of the obvious speed advantage of the CRT there is often a need to review a previous question or answer. If all reviewing, however brief, requires a query instead of a glance up the page, the responder must lose his train of thought.

a. The Question as a Unit of Programming

Programs in CEL are written in units called questions. As the name implies, a question is all the coding concerned with eliciting an item of information or a response to one interrogation question. There may be more than one conversational exchange in a question, but there will normally be only one data base element elicited. This may be an array of elements but would not normally be a set of dissimilar items. The question consists of a heading, text to be displayed by the computer, elicitation and analysis of a response, and an ending. A question as a unit of programming should only be entered at the beginning, or heading, of a question, not its interior, although no PL/I restriction requires this. There are labels at the beginning of a question and at several points within one. The use of the latter is described below.

b. Subroutines

Let us consider a set of subroutines which perform some of the basic tasks in the use of CEL. There are six major ones, concerned with message transmission, response elicitation, branching and program bookkeeping.

ASK assumes a message is stored in data item MSG. It displays the message on the terminal indicated and elicits a response. The response is stored in item RES. The program author, then, deals only with MSG and RES and need not be concerned with the mechanics of message transmission or receipt.

TELL causes the contents of MSG to be displayed, but elicits no response.

INIT is used in the heading of a question to initialize a number of parameters used in program bookkeeping.

NEXT is used in the ending of a question to perform bookkeeping functions and to branch to the next question to be used. At many points within the question the decision on what question to branch to next might be made, but rather than performing the transfer, a variable is set to the address to which transfer is made later. Actual branching from one question to another is always done from NEXT.

UNRECOG counts the number of successive unrecognizable responses, provides a message to the console after each and, if necessary, terminates the conversation. Provision is made for having a different machine statement follow each successive unrecognizable response.

KEYEXT extracts key words and/or punctuation from the text of a user's response. The punctuation option is often used, even when short answers are elicited, to remove the differences among, for example, yes, YES, and Yes. Keyword extraction is used for indexing responses for later retrieval, and for testing the validity of a text response in an instructional mode.

A series of retrieval subroutines permits initial storage, replacement, retrieval or deletion of an item or a structure in the program data base on disk.

c. An Example

To write a question, the program author must have: a label, a CALL to subroutine INIT and a CALL to subroutine ASK. Prior to ASK, he must have set the character string designated by the label MSG equal to whatever message or question he wishes to present.

Response analysis consists largely of a series of IF statements, each of which may have a THEN and an ELSE clause following. Figure 1 shows an example from an instruction executive program. In reading this condensation of a program, bear in mind that a branch command would normally occur within each DO group.

```
LABEL:  CALL INIT;
        MSG = 'IN WHAT YEAR DID THE CIVIL WAR BEGIN?';
        CALL ASK;
        IF RES = '1865' THEN DO; ... END;
                        ELSE DO; ... END;
        IF RES = '1861' THEN DO; ... END;
        CALL UNRECOG;
```

Figure 1. A Sample Interrogation Program

Here, the program author anticipates responses of 1865 and 1861. Any other reply by a responder is unrecognizable. The author could, of course, have anticipated a larger number of answers, or he could have used inequality matching and completely precluded the possibility of an unrecognizable answer. Here, the author may specify a set of commands to be executed if the answer is correct, which may include a message to the responder, the computation of a grade or the setting up of a switch for a later branch to a new question. In the example, the author has chosen to include a clause to be used in the event the correct answer is not given, regardless of what other answers are given. This is the ELSE following the 1865 response test. Here again, he may pass a message, compute a grade, prepare for a transfer, or perform any other function.

If the student's answer is unrecognizable, the subroutine types out a message informing him of this. Some authors may choose, at this point, to review the mechanics of answering, assuming the cause of the error might have been a keying error. With the second unrecognizable answer, the author might give a hint, and with the third, he might warn that the responder will be cut off if another unrecognizable reply is given.

Within the DO ... END group, any assignment statements or subroutine calls are permitted. The IF statements may be complex, allowing nested logical expressions such as IF(A = '2') & (B = '3') | (C = '4') THEN .... PL/I allows for another IF in the DO ... END group, but CEL does not permit this. In PL/I notation & implies AND and | means OR.

d. Labels

Labels on questions must be included in a label array. We use the form LL(n). The variable n has many uses within a program. For one, it becomes the means of communicating transfer information. If, within an IF statement, it is decided that the next question to be asked is question LL(X), a direct branch cannot be made at this point. This, it will be recalled, can only be done from NEXT. Instead, an index, q, is set to the value X and later, in NEXT, transfer is made to LL(q). Internal labels within a question are explained in the section on instruction.

One function of INIT and NEXT is to maintain a list of questions executed. This list, which need not contain more than about the last five questions executed, is used when the responder calls for BACK. Typically, he would do this if he realized he had given the wrong answer to a question and wants the chance to answer again. By responding BACK to the next question asked of him, he backs up to the previous question. When that is asked, he can answer responsively or BACK again, until he has found the question he wants or has reached the end of the list. BACKing also requires the removal of data base elements that were inserted as a result of the wrong (but legal)

answer.

4. Response Processing. Almost anywhere within a program an author may enter processing statements. Usually, these will be restricted to use just before the response is elicited or in the DO groups following the string of IFs in response analysis. In a program produced through the Generator (Section III.2) there are more restrictions on placement.

Processing can consist of any assignment statements or CALLs to subroutines. While a question is only intended to elicit one information element, the author may include instructions for several messages to be displayed to the responder within one question unit, giving him instructions, congratulating him (if a student) on a correct answer, and so on.

Figure 2 shows a short example of a question to illustrate a typical response analysis.

The following comments are keyed to line numbers in the figure.

1. The question label is LL(1). The first function is initialization.

2,3 The first question is set up and asked.

4. Function MEMBER (A,B) returns '1' if A is a member of array B, '0' otherwise. If the name is known to the roster:

5. Set BRL (branch label) to 2 to indicate that the next question to be executed is LL(2).

6. Now branch, within the question, to LE(1) (E for ending).

8. This begins an ELSE clause, used when the name is not on the roster.

9. This illustrates use of the data base in composing a message. The message being formed here includes the name just entered, concatenated (||) with the text shown.

10. This message is displayed, but calls for no response.

11. LL(99) is assumed to be a sign-off point.

14. The label LE(1) indicates the ending section of the question.

```
LL(1):    CALL INIT;
          MSG = 'WHAT IS YOUR NAME?';
          CALL ASK;
          IF MEMBER (RES, ROSTER) THEN DO;
                BRL = 2;
                GO TO LE(1);
                END;
                                    ELSE DO;
                MSG = RES||'IS NOT ON ROSTER.  PLEASE CHECK
                                WITH PROCTOR.';
                CALL TELL;
                BRL = 99;
                GO TO LE(1);
                END;
LE(1):  CALL NEXT;
```

Figure 2. Response Processing

5. Review and Update. Whether in an instructional mode or an interrogation mode, there is a need to be able, at any given point, p, in a program, to execute selected questions and then return to p. In interrogation, we do this in order to change the contents of a data base item. Recall that our method of updating (Volume 1, Section II.5) requires that questions concerning the item to be changed be asked again and that a different path through a set of related questions might result from the change than was originally used. In instruction, at various points in a course the author may wish to compute a set of review questions based upon the individual student's performance. Again, given a set of questions to be reviewed, the execution path through these questions may be different than the original path through them.

The reviewed portions may consist of a single sequence of questions, or it may be an unconnected set of questions. Presumably, the number of possible combinations is large or the author could have set up the subroutines himself. Instead, he programs criteria for selecting review questions as a function of performance. In interrogation the same technique is used for updating previously entered documentation. We have used the principle that programmers make changes to documentation by identifying the information element to be changed and then being reinterrogated about that element.

Reinterrogation or review might result in execution of a different sequence of questions than were originally executed. Hence, what the author specifies is not the actual sequence of review questions but the beginning and end points of a set of questions. Questions are then executed in normal sequence until the specified end point is reached and return is made to the sequence control point. Any number of such segments can be specified by an author to satisfy any one review or reinterrogation item.

At the end of each question, then, a test must be made to see if the question was executed in sequential or review mode. If sequential, the next question executed is that indicated by the question's branch logic which points to the next question to be asked. If in a review mode, a branch is made out to the sequence control routine which checks to see if this question marks the end of a review sequence. If this is so, control passes to the next review sequence or to the elicitation of a new updating or review command. If it is not the end of a sequence, control goes back to the NEXT subroutine and that routine branches to the question indicated by BRL, a variable which is set to the index value of the label to which it is desired to branch.

6. Special Responses. The responder at a terminal can alter the sequence of operation of an executive program and may interrupt its operation and resume later. Whenever his keyboard is unlocked the responder may give one of the responses listed below instead of the reply requested. In experimental work these responses have been prefaced with // so as not to be confused with valid replies.

GO TO n causes the executive program to begin executing question LL(n) and picking up future sequence commands from there.

QUERY invokes an on-line retrieval system. The responder may then query the data base on an information element number (IEN) or a key word. An IEN is assigned to each element in the data base. If a key word search is elected, the program retrieves the list of IEN's in which the given word has occurred and then executes a series of IEN searches. Upon completion of the search, the executive program resumes with the question it tried to execute when interrupted.

REPORT invokes a report generating program. As presently written, this program gives the responder the option of receiving a standard, pre-designed report or designing his own report by specifying a set of IEN's. The report, in either case, is printed on the high speed printer and then the interrupted question is resumed.

BACK causes the executive program to return control to the previous question executed. A short history of the last n questions is maintained to allow backing over several questions.

CHANGE indicates that the responder wants to stop working on the present UOP and switch to a different one--perhaps to describe a new one he has just named for the first time.

SIGN OFF terminates the executive program. The responder is informed of the question number at which he terminated and he may use this to resume in his next session.

The following responses are under author control but are usually used as indicated.

NO signifies the responder does not wish to answer the question. This is interpreted as meaning that no answer is currently available or that the question does not apply. This response is often used in design interrogations when full details are not yet known.

END is used to signify the end of a list when an array of responses has been elicited. A question is asked and the responder gives one array element. After each element the keyboard is briefly locked as the reply is stored. When the keyboard is unlocked, the responder enters the next element or //END.

HELP in effect means "Give me a restatement of the question." When displaying to the relatively slow typewriter, program authors may use terse wording to avoid boring the responder. The responder, however, may then occasionally need amplification of the question. The author, of course, must have anticipated this and provided more than one version of the question. This function has not been implemented yet.

HINT might be used to ask for assistance in answering a question in an instruction program.

7. Documenting CEL Programs. Interrogation and instruction programs require documentation, just as do other forms of computer programs. The structure of a CEL program is generally far simpler than a typical computer program. The major point of interest is what is said to the responder or terminal user and what answers are anticipated from him. Hence, a different, fully automatic form of documentation has been devised for these programs.

Briefly, the documentation of an instruction course will consist of a list of questions, the anticipated answers and a summary of what is done if a given answer is received from the terminal. This listing is in order of question number. The listing is followed by a key word index of all text of machine utterances.

Sample output of a preliminary version of this documentation is shown in Figure 3.

QUESTION NUMBER: 001

THE BINARY NUMBER SYSTEM.

THE BINARY NUMBER SYSTEM CONSISTS OF TWO DIGITS, A ZERO AND A ONE. THESE USUALLY REPRESENT THE OFF AND ON STATES OF A BI-STABLE ELEMENT. BY COMBINING THESE DIGITS IN STRING FORM, NUMBERS OF INCREASING SIZE CAN BE REPRESENTED.

SINCE ONLY TWO DIGITS ARE USED, NOT ALL NUMBERS CAN BE REPRESENTED. TRUE OR FALSE?

ANSWER	BRANCH TO
TRUE	001
FALSE	002

QUESTION NUMBER: 002

EACH PLACE IN A BINARY NUMBER REPRESENTS A POWER OF TWO, WITH THE RIGHT-MOST BEING TWO TO THE POWER ZERO.

THE BINARY NUMBER 1010 IS THE REPRESENTATION OF A DECIMAL:

	ANSWER	BRANCH TO
A: ONE THOUSAND TEN	A	002
B: FOUR	B	002
C: TEN	C	003
D: ONE HUNDRED ONE	D	002
E: NONE OF THESE	E	001

Figure 3. Example of CEL Program Documentation.

KEY WORD INDEX

BINARY	INS 001	INS 002
BI-STA	INS 001	
COMBIN	INS 001	
DECIMA	INS 002	
DIGITS	INS 001	
EACH	INS 002	
ELEME	INS 001	
FALSE	INS 001	
FORM	INS 001	
HUNDRE	INS 002	
NONE	INS 002	
PLACE	INS 002	
POWER	INS 002	
SIZE	INS 001	
STATES	INS 001	
STRING	INS 001	
SYSTEM	INS 001	
TEN	INS 002	
THOUSA	INS 002	
TRUE	INS 001	
USUALL	INS 001	
ZERO	INS 001	INS 002
1010	INS 002	

Figure 3. Example of CEL Program Documentation (concluded)

II

TECHNIQUES OF INTERROGATION

1. The Data Structure. From a programming point of view, the object of an interrogation is to make entries in, or make changes to, a data structure. This structure may be an image of a report, the raw material from which a report is made up, a computer file to be used only as input to another computer program, or, as we shall describe in Section III, a computer program, itself. The first step in creating an interrogation program is to design the data structure that the program will work with.

The requirements of the structure that CAINT is designed to work with are that each element must have a unique information element number, and that higher order elements (sets of lower order elements) are possible and will have IEN's that reflect this hierarchic relationship. Each item of information which can be independently elicited or changed should have a separate IEN. The nature of the information retrieval system implemented for the experimental versions of CAINT requires that retrieval be done on the basis of IEN's. Hence, any information in the data base that is to be stored on, and retrieved from, disk memory must be included in the IEN structure.

We have pointed out that any information in the data base may be used within any question, to contribute to decision-making, to assemble the text of a question, or as part of response processing or student performance analysis. In any given question, then, any number of information elements may be retrieved, but only one should be stored. This information, such as various program switches, constants, etc., may be defined via PL/I DECLARE statements at the discretion of the programmer.

In setting up an IEN structure, the major rule is that every element or set of elements must have a unique IEN. For example, if there is to be an element called Name of Input, we might assign it IEN 1.1. If we then wish to allow for more than one input, making the Name item an array, we must assign each element of the array an IEN and the array, itself, an IEN. The array, then, might inherit the number 1.1 and the first element in it becomes 1.1.1, the second element 1.1.2, etc. We permit the use of the notation 1.1.n to denote the nth element of an array. If we wish to further extend the array, by listing name and source of each input, we would require the following structure:

1	INPUT
1.1	ARRAY OF INPUT NAMES AND SOURCES
1.1.n	NAME AND SOURCE OF INPUT N

1.1.n.1 NAME OF INPUT N

1.1.n.2 SOURCE OF INPUT N

In the interrogation program reference may be made to IEN 1.1.3.2, which is the source of the third input. It should be apparent that a change in the file structure may induce a change in the interrogation program and vice versa. It should also be apparent why indiscriminate changes cannot be made in data base structure without consultation among all programmers of CEL programs using the affected data items. Mention was made of this restriction in Volume 2, Section II.2.

2. Interrogation Sequence. Having decided upon the data structure to be used, the CEL programmer must then consider what questions to ask, under what conditions to ask them, and in what order to ask them. The first of these considerations is largely one of how to communicate with the responder and we shall discuss it in greater detail in Section II.3.

Not all questions are used in every interrogation. As a simple example, if a programmer taking an interrogation has just replied that a particular data item is alphabetic, we do not ask him whether he will use fixed or floating point representation. Each response received should be used to the greatest possible extent to by-pass irrelevant questions and to make relevant ones more specific.

There is no requirement that the sequence of interrogation, or presentation of questions, follow the sequence of storage of information in the IEN structure. A CEL programmer may choose to follow one item to its completion, say to get all information about one UOP before proceeding to the next, or he may choose to get a little information about each UOP before getting the detail on any of them. He may elect to skip around or to allow the responder to make the choice as to topic covered, insuring that some record is kept available to the program of what IEN's are covered and which ones remain to be completed.

3. Considering the Responder. Writing an interrogation program is carrying on a conversation vicariously. Since it is the object of this conversation to acquire a specific, complex set of information, the job is by no means trivial. The CEL programmer must consider what the responder knows when he takes an interrogation, how fast he is probably able to work, how fast he would like to work, and how easily he can be bored. Skill in CEL programming does not come automatically from skill at conventional programming. It requires a mixture of conventional programming skill, skill in writing or speaking--the communication of ideas to people--and experience in both writing and taking interrogations.

a. The Scope of the Question

For reasons having to do with initialization of programs, we have suggested restricting a question to the acquisition of a single information element. In practice, a CEL programmer must consider carefully how much information to put into a single IEN and must consider the effect on the responder of asking either too much or too little at one time.

If too little is elicited, the responder will become bored and will lose the train of his thought. If, for example, a date were elicited by three questions asking, 'DAY _____', 'MONTH _____', 'YEAR _____', the responder may be expected to show irritation at the slow pace. If, on the other hand, the program asks, "Give all twelve information items about each of the inputs you have mentioned." the responder may well forget some of them and lose his place in the sequence. The ideal interrogation program will help the responder to remember, by cueing him occasionally, will recognize that the responder is learning technique and can improve his performance with practice, yet will see to it that each question is answered before going on to the next.

To assist the CEL programmer we have developed a question modification technique and have considered use of a second one, not yet implemented. The first is based on the programmed instruction concept of a fading cue, [1] which, in our context, means reducing the amount of explanation given with each successive use of a question. Recall that in interrogation programs, the responder is going to see the same basic set of questions over and over again, as he reports on different UOP's and data items. The first time, or the first few times, he sees a question he needs an explanation of what it means, and of where the answers fit in the overall data base and reports. After a few times, particularly when using a typewriter terminal with its slow output rate, the responder needs only to be reminded of what question is being asked, not necessarily given the full statement of the question.

At present, we allow for three different versions of a question and shift from version one to two and then to three after a predetermined number of uses. After a programmer has answered the question, "What is the function of this UOP?" several times, he need only be asked, "Function" to know how to respond. Eventually, we would allow a responder to recall version one in case he forgets its meaning, by responding //HELP to the short form of the question.

The second technique is to enable the CEL programmer to combine several questions into one, in the manner of a fading cue, based upon repetition of the questions. A succession of questions might be these:

First iteration:	WHAT IS YOUR NAME?
	WHAT IS YOUR RANK?
	WHAT IS YOUR SERIAL NUMBER?
Second iteration:	GIVE YOUR NAME, RANK, AND SERIAL NUMBER SEPARATED BY /
Third iteration:	NAME/RANK/SERIAL NO.

In any updating activity, a return to the separate questions is required.

b. Reaction Times

Much is said and written on the subject of response times in man-machine systems, although little has been firmly established. We have not done any formal experimentation in this area, but we have developed some convictions. Not surprisingly, we find that responders react differently to delays at different points in a conversation. We may classify these approximately as follows:

(1) Within a question. As a function of both programming and hardware, some systems impose delays, or otherwise encumber the responder, during the printing of a machine utterance or the entry of data by the man. For example, the 1440 CAT system with which we did our early experimentation imposed a fixed, one-second delay at the end of any line of print, even if in the middle of a sentence. We found this to be disconcerting to the reader. Some versions of the same system require the responder to depress a request key each time he wishes to make a response, then wait a variable length of time, rarely exceeding about two seconds, for the keyboard to be unlocked. This is also bothersome because it delays the man just at the point where he knows what he wants to do or say and is ready, but finds the machine not ready. The conclusion is that these delays are least tolerable of all--mechanical delays that interrupt the logical process.

(2) After a service message. A service message is one that asks a procedural or administrative question, not a substantive one concerned with the data base. It might ask, "What subject do you want to consider next?" We use the term service question by analogy with the term service traffic in communications, referring to messages about the operation of the communications network over which sent. These questions have little intellectual content and require little thought by the responder. He would not like to be delayed by such a message or by the process of having that message acted upon.

(3) After a substantive response. When a substantive question has been asked and the answer given, a unit of work has been performed. The responder now knows the subject

matter will change, at least slightly, and he is prepared to accept a small delay. It may be necessary because it is likely that some data has to be stored at this point, and the program may impose a delay to do so.

(4) After a short query. When the responder asks the computer a question, which he feels is a simple one, (such as //HELP, or "retrieve TEN 1.2.3") he expects quick response, delay possibly in terms of seconds, but not minutes. These are short, precise questions. He knows what the answers will look like and needs only the content.

(5) After an extensive query. When, on the other hand, a question is asked that is of an analytic nature, such as, "What programs make use of file A for input and do not have an error routine?" the requestor is probably willing to tolerate a lengthy delay, even of several minutes. Here, he does not anticipate the answer. He does not know if there is an answer. He knows he will have to wait before he can proceed. We can also include other requests in this category, such as requests for generation of a report.

c. User Control of the Interrogation

The CEL programmer can delegate a great deal of control over an interrogation to the responder, something he will be more inclined to do if he knows he is working with experienced people. All the user control techniques have been described elsewhere, but we shall briefly review them here.

(1) Updating. Once a responder establishes a file on his major topic (program or data file) he can operate entirely in an updating mode thereafter. This means he specifies what topics he wishes to be be interrogated upon and he restricts the conversation to these items only.

(2) The CHANGE response. At any time during a program logic description, the responder can change the UOP he is describing by answering //CHANGE to any question. He then provides the name of the UOP he wants to switch to. In this way, he is free to document UOP's in any sequence that is meaningful to him, unconstrained by what the CEL programmer may have thought best. The CEL programmer may, however, disallow this response.

(3) The GO TO response. This response has the same effect as if it were a program statement. It causes a transfer to the named question in the interrogation program, but is available mainly for debugging purposes.

(4) The BACK response. This is another way the responder can transfer to a different question, in this case to any question of the last n previously asked.

(5) The QUERY and REPORT responses. These enable a responder to interrupt the interrogation to ask for information from the data base. Data will be returned at the terminal in response to a QUERY or at the line printer in response to a REPORT.

III

INSTRUCTION IN ESDP

1. Objective. A major objective of ESDP has been to include an integrated instructional subsystem. Briefly, the rationale behind this objective is that large programming projects always have problems caused by turnover of personnel and expansion, both in the programming ranks and in the ranks of the managers and operators of the resulting system. It is rare for a programming project to have a comprehensive training program for new personnel that is addressed to the system rather than to the mechanics of programming the computer or to operating some component. It is rarer for such a curriculum to be both effective and up to date. The instructional system described here is one which can satisfy these requirements, yet be relatively inexpensive to operate and maintain.

Specifically, the objectives of the ESDP instructional system are:

a. To enable the transformation of ESDP-acquired documentation into effective instructional text, both rapidly and easily.

b. To enable instructors (authors of instructional text) to prepare original training material not copied from existing text, rapidly and easily, when either the requisite documentation does not exist, or, in the opinion of the instructor, it is not of instructional quality.

c. To enable instructors to prepare either computer assisted instruction or programmed instruction courses (i.e., instruction presented in printed form, off-line) at their option.

In 1966, before the start of the present contract, IBM conducted a short experiment in automatic production of instructional text from ESDP documentation. The approach taken was to embed the documentation in a standard format that would successively present the student, through a terminal device, with a topic title (taken from the headings in an ESDP report), the text supplied by the programmer in documenting his program, a standard question calling for the student to summarize what he had just read, then a summary of the previous text supplied by the original programmer-documentor. (Summaries of topics were and still are considered a useful documentation device for ESDP documents.) The student was then asked to judge whether he answered correctly, that is, if his summary substantially agreed with the programmer's summary. If not, he was asked to explain the discrepancy. His answers and explanations would be subject to review by an instructor at a later time. The student was then given the next topic in sequence and the process repeated. At a number of different points in the presentation of material and his response to it, the student could ask for a review of any

previously covered material. Thus, if he had forgotten the definition of a data item previously explained, he could, before answering the current question, ask for a review of the item definition. The student was able to select his own path through the instructional material on the basis of major subdivisions. If there were several major subjects, he could select the order of considering them, but within a major subject he had to follow the prescribed order of presentation of minor topics.

A computer program to generate such instruction programs was written and made operational. The results were found to be unsatisfactory. Mainly, this was because the student was left on his own to judge his ability to recognize and interpret the important concepts in a paragraph with the result that effective feedback to the student was not provided. One major change was required: the ability to tailor the question to the text, and to enable this to be done without the course author or instructor having to completely retype or regenerate the documentation or to become too entangled in computer programming. Other requirements to be levied on a new instruction generator were the ability to generate a program that would, without modification, reflect minoor changes in the source documentation, and the eventual interconnection of the generator with the ESDP data base to allow students to ask information retrieval queries of that data base while taking instruction.

An instruction generator that can test these concepts has been designed and a preliminary model made operational. The assumptions underlying this design are the following:

a. In most cases ESDP documentation will exist before the instruction course is written or compiled and most of this documentation will be used as text material in the course.

b. The courses will be assembled by people familiar with the object program system, programming in general, and instructional technique. It is not expected that each object system programmer will compile his own training materials. While much mechanical assistance will be provided to the instructors, there will be no diminution of the requirement for instructional skill on their parts.

c. The object system programs that form the basis of the instruction can be expected to change; therefore, the problem of keeping the course up to date will always be present.

2. <u>An Instruction Generating Program</u>. Two models of an instruction generator have been designed and the first of these has been made operational. The purpose of the first model program was to test the feasibility of the approach and to serve as a test vehicle for future experiments.

a. The Initial Model

The program has the following features and limitations:

(1) The generator is an interrogation program written in the CAINT Executive Language. The object program (resultant instructional program) is in the same language. The object program is generated through an interactive process in which the conversation between the generator and instructor is in near natural language. Little programming skill is required of him, but the facility is available for those who are skilled to enter PL/I statements of their own into the generated instruction program.

(2) If an ESDP data base is available, the generator can make use of it. An instructor can incorporate any portion of the base into his program by identifying that portion; he need not copy it. If such data is not available, or if there is any segment of it the instructor does not choose to use, he may compose his own text.

(3) Only multiple choice questions can be used (in Model 1). A question is identified as being true/false, yes/no or other multiple choice, and the student is then restricted to answering true or false, yes or no, or a, b, c, d, ..., j. Any answer other than one of these anticipated answers is unrecognizable.

(4) An instructor may insert PL/I language statements to analyze or process responses, but he is able to compile a fully operable and meaningful course without recourse to any programming whatever. The generator assists in computing a student's grade, and counting unrecognizable answers. It can be made to store responses and type out special messages, all requiring of the instructor only that he specify what is to be done, not how.

(5) The course checks for unrecognizable answers and offers the student three chances to change an unrecognizable reply to a recognizable one. If, after three tries, an answer is still unrecognizable, the course cuts the student off the computer. The course also keeps track of student progress and allows him to log on and off, resuming where he left off.

b. The Second Model

A second model of the program would expand on the basic system, primarily enlarging the number of possible answer types and corresponding response analyses. It would provide the following features beyond those offered in the first model.

(1) In addition to multiple choice, instructors could allow for:

- o item responses (one or more distinct fields of information in the reply)
- o phrase response (short, natural language replies)
- o text responses (long natural language replies)
- o lists of item responses.

Each question may be of a different type.

(2) Key word extraction routines will be available for call by the instructor in conjunction with phrase or text responses. These enable him to make some checks on extent of subject coverage in a response, (e.g., did the student mention "end of file,") but still, of course, does not judge the adequacy of the content of the reply.

(3) The course author will be given more response checking alternatives since a response no longer would be required to match exactly one of a limited set of stored answers, as in multiple choice. Instructors would be offered range checks, absolute value checks, inequalities, exact equality, etc. Also, they could provide their own response processing, before and after response testing, to allow for far more sophisticated response analysis than can be explicitly anticipated in design of the generator. For example, an instructor can make use of a combination of the student's performance (grade) thus far in the course and his latest answer to decide what to do if that answer has particular characteristics.

(4) The capability to back up, or retrace a path will be provided when the author replies BACK to a question.

(5) With Model 1 there are several subroutines available for author use. These, for example, enable him to program transmission of a message to a student by entering only message text as a parameter. The later model will enable the course author to "write" his own subroutines which may then be called anywhere in his program. For example, he may wish to use a text analysis routine of his own design. He can indicate to the Generator that he wishes to compose a subroutine, name it, compose it, then assume it is available for his use whenever convenient.

(6) Model 2 will have a full capability to handle program changes after the course has been generated or partially generated. This will be done by having the author specify the operation (add, change, delete) and identify the location within the course. He then resumes the course-generating interrogation to supply new coding.

A generator program capable of meeting these specifications should be capable of generating itself. This would be a significant milestone, because the generator is, itself, a CAINT interrogation program, and if it can be made self-generating, then it should be able to generate any interrogation program as well as any instruction program. Even solely within the context of instructional systems, this is significant because instructors may wish to have special information elements, just for instructional use, elicited during the original documentation interrogation of programmers. Then, when changes are to be made in the ESDP data base or in the manner of interrogation, these can be accomplished and integrated with instruction more easily.

c. Handling Changes in Documentation

An option with either version of the generator program will be to allow the instructor to make use of documentation provided by the programmer who wrote a program without actually copying the text. The instructor will identify the text by its IEN, and the generator will respond by placing appropriate PL/I statements to retrieve the text into the object program at object time. In this way, documentation changes made by the programmer would be reflected in the instruction course without additional effort on the part of the instructor. The instructor retains the option of not using the programmer's text, for whatever reason, and instead supplying his own text or a paraphrase of the original.

This technique handles one aspect of changing documentation. Through its use, instruction programs automatically adjust to small changes made in documentation without, for example, invalidating a course because the programmer changed the spelling of a word or amplified an explanation. Another aspect of changing documentation is found when the originating programmer changes the structure of the documentation for a program. This may imply a change in structure of the program being described, as well. For example, the programmer may add a new subroutine, or divide an existing procedure into two procedures, or simply make use of an additional data item in an existing calculation. Such changes do not simply cause a replacement of an existing documentation item by a new one, but cause additions or deletions to arrays of program elements, or possibly change the relationship among existing elements of an array. A course generated through the techniques discussed here cannot automatically adapt to this form of change. When major documentation changes are made, ESDP has the capability to store the fact of this change and, when an instruction course is to be taken, make known to the student the fact that the course is out of date. In a large number of cases, this will make little difference, for one change in the subroutine structure or introduction of another data item will generally have little impact on the student's understanding of the basic concepts of the program and its overall structure. A regular program of review and revision of generated courses can

keep the complete instructional system reasonably up to date.

Both instructors and students must be aware that an instructional system is different from an information retrieval system; they are not interchangeable with each other. Instruction concentrates on manner of presentation and attempts to ensure that its users gain a good general understanding. Retrieval has the mission to provide up to date information, and may presume that its user can manipulate the system and interpret its output with skill.

Another approach to automatic updating of instructional material is to hold the programmer who writes the documentation, or changes it, responsible for updating the instruction. While this approach would work well mechanically, it requires only a few more questions to be asked of the programmer as he makes his documentation changes, we do not feel that programmers in general are necessarily qualified instructors. Skill in writing and in instruction is required.

d. Production of Programmed Instruction Courses

The current experiment in producing instructional text is not concerned with programmed instruction (PI), but an example is offered below to show that the ability to produce CAI courses by computer can easily yield PI courses as well.

A typical question, or unit of instruction, generated by the ESDP instruction generator, would have approximately the form shown in Figure 4.

This is an abbreviated version of the coding for a question, but shows most of the essential elements. The same question and branching decisions could be used in a PI format, which might be as shown in Figure 5.

The only real difference between this PI format and a CAI format is that all "processing" is done by messages, not by arithmetic. The same branching techniques are used, the same ability, during generation, to retrieve IEN text or have the instructor supply it is needed. The major difference is really that PI courses are printed directly, not compiled as a computer program. We anticipate that use of the PI option in connection with the ESDP instruction generator would produce a program which would, in turn, produce the PI text when the student is ready to take the course. That is, just as with CAI, the instructor prepares an object program, but this object program is compiled and run to produce PI text only when the student is actually ready to take the course in order that all late changes in documentation be automatically incorporated in the printed text.

```
INS(1):   *MSG = 'IS THE PERSONNEL FILE SEQUENCED BY NAME?';

          CALL ASK;

          IF RES = 'YES' THEN DO; GRADE(1) = 1;

                MSG = 'GOOD'; CALL TELL; BRL = 2;
                GO TO INS(1); END

          IF RES = 'NO' THEN DO; GRADE(1) = -1;
                MSG = 'WRONG, TRY AGAIN'; CALL TELL:
                GO TO INS(1); END
```

* The ASK/TELL series of subroutines assumes the text of the message to be displayed is in data item MSG. If a response is called for (it will be in the ASK subroutine if used) the response will be in RES.

Figure 4. A Question in CAI Form.

```
INS(1)    Is the Personnel File sequenced by NAME?

          If your answer is YES, go to INS(2)

          If your answer is NO, go to INS(3)

INS(2)    Good, the correct answer is YES.
          Now, the next question ...

INS(3)    Wrong.  To review, we said earlier
            (text of earlier IEN)

          Now go on to INS(2)
```

Figure 5. A Question in PI Form.

IV

INSTRUCTIONAL PROGRAMS

1. The Program Model. The basic model of the instruction program is that of a branching type program, the creation of which is generally attributed to Norman Crowder.[2,3] The other major programmed instruction model, a linear or Skinnerian model, [3,4] is included as a subset within the overall model. A linear model is one which always branches to the next instructional frame or question, regardless of student response, while a branching program has the option of going to a diffferent successor for each different recognizable student response. In practice, the branching type programs are usually multiple choice, while the linear programs use a constructed response where the student must compose the answer, rather than having several possibilities presented to him for his consideration. One advantage of computer assisted instruction is that a great many constructed response answers can be anticipated by the instructor, so that the branching technique may be combined with the constructed response technique, giving greater flexibility to the course author and more feedback to the student.

We define the basic unit of an instruction program to be a question or unit of instruction. The term question is slightly ambiguous, but is handier to use. Within the question (used here as a synonym for unit of instruction) there can be a question (used here as an interrogatory sentence). The unit of instruction consists of the elements listed below, some of which may be omitted. This is a slightly more rigid organization than specified for CEL programs in general, in Section I.

a. Heading

This is a call to a subroutine provided by the generator. Generally, the instructor does not control the heading. The subroutine called handles certain internal "housekeeping" details needed by the instructional program.

b. Title

At the option of the instructor, the UOI or question may have a title, which may be the title of the UOP being described or any other title provided by the instructor.

c. Text

Again at the option of the instructor, there may be a text portion which would normally be used to present the documentation text provided by the programmer when he documented his program. Alternatively, the text can be provided by the instructor. The text part of a question is displayed to the student but does not elicit a response from him.

d. Question

This part of the UOI asks the student something about the text he has just read. Normally, there need be no sharp break between text and question in programmed instruction. We draw this distinction because often the text will be a copy of text supplied by someone other than the instructor, while the question will be supplied by the instructor. If the instructor were composing his own text and questions, he need not make the distinction, and could treat the question as containing the text as well. A question elicits a response from the student. Its use is optional. If not used, only the title and text are displayed to the student and the course proceeds as programmed, without considering any response.

e. Response Analysis

This portion of the UOI tests the student's response for compliance with predicted responses or functions of responses or on other data items, stores the student response and decides where in the course to branch next.

f. Ending

The ending is analogous to the heading in that much of it is supplied automatically by the generator program and is used for program housekeeping activities. It may also contain processing steps to be performed regardless of what response the student gave to the question, and hence is separated from the response analysis section. For example, a branching decision could be made on the basis of previous response patterns or total score.

2. An Example. Figure 6 shows an example of the coding of a question in the object instructional program. Labels shown on coding are labels that might actually appear in the PL/I coding.

```
INS(1):  CALL INIT;

         MSG = 'INPUT';
         CALL TELL;

XT(1):   CALL RETRV(IEN);
         CALL TELL;
XQ(1):   MSG = 'WHICH OF THESE FILES IS UPDATED DAILY?
                            A  PERSONNEL
                            B  PAYROLL
                            C  NEITHER OF THESE';
XA(1):   CALL ASK;

         IF RES = 'A' THEN DO; GRADE(1) = 1;
                               MSG = 'GOOD';
                               CALL TELL;
                               BRL = 2;
                               GO TO XE(1);
                               END;

         IF RES = 'B' THEN DO; GRADE(1) = -1;
                               B(1) = B(1) + 1;
                               MSG = 'CAREFUL...TRY AGAIN';
                               GO TO XA(1);
                               END;

         IF RES = 'C' THEN DO; GRADE(1) = -1;
                               MSG = 'READ THE TEXT AGAIN,
                                          CAREFULLY';
                               CALL TELL;
                               GO TO XT(1);
                               END;

         CALL UNRECOG;
XE(1):   CALL NEXT;
```

Figure 6. A Unit of Instruction or Question.

The following comments are keyed to line numbers in Figure 6.

1 The label of the question is automatically composed, and fits the requirements of a label in a PL/I label array. A CALL is inserted to the standard bookkeeping routine.

2,3 The title "INPUT" is displayed. This title was composed by the instructor, or course author; otherwise, a retrieval call would be here that would retrieve the title from a list in memory. The subroutine TELL does not call for a response.

4 The label here is an internal label, internal to a question, and has the same subscript as the main question label. The "T" denotes start of the text presentation. The call to subroutine RETRV will cause the text stored under that given IEN to be retrieved and stored in data item MSG.

6-9 Here the question is being posed. The text has been supplied by the instructor, in the form of a multiple choice question.

10 The label denotes the beginning of the response acquisition and analysis section. The ASK subroutine elicits a response from the student and places that response in data item RES.

11 The first response checked for is A; this being automatic if a multiple choice question form has been selected. If this answer has been given, a grade for the question is assigned. The value of the grade is determined by asking the instructor during the generation of the course whether the answer is correct or not. If correct, the student gets +1, if not, -1. It is not necessary that each question be identified as either right or wrong.

12, 13 The instructor has chosen to "reward" the student with the comment "GOOD."

14 The instructor has stated that he wants to go next to question 2, which is at label INS(2). Branching out of the question is done only at the end of the question. Here, the value of BRL (branch label) which will be used to index a label variable is set to 2.

15 Transfer is now made to the ending routine for the question, XE(1).

17 "B" is a wrong answer, hence the grade is set to -1.

18 The instructor wants to know how often this answer was given, so he introduces his own counting variable, B(1).

19,20 The student is told to answer the question again, and is branched to the beginning of the response analysis area. The question will not be repeated, but the branch is to a CALL ASK statement.

25 Here, in the analysis of the third recognizable answer, the instructor feels the student must have grossly misunderstood the text to have given this answer, so he is going to force a review by repeating the text by a branch to the beginning of the text presentation area, XT(1).

27 If no recognizable answer is given, a call is made to subroutine UNRECOG which will converse with the student, try to get a recognizable answer from him and, if it cannot, cut the student off.

28 The label XE(1) denotes the ending of the UOI. The subroutine NEXT performs the actual branching to the next question.

3. Composition of a Question. More specifically, the functions and options available for each component of a question are:

a. Heading

All questions must have a heading, hence the heading is not under control of the instructor. Included are the label for the question (the label of the PL/I code group associated with the question). All such labels are part of a PL/I label array and consist of an alphabetic prefix, INS, followed by a subscript generated by the generator program. Next, the documentation to which the question pertains is ascertained. This is elicited from the instructor, as the composition of each question starts, by the computer asking him what IEN his question is associated with. Finally, the heading concludes with a call to the INIT(ializing) subroutine which records, for the object program, which label is being executed and resets various counters and registers.

b. Title

The title is entirely under control of the instructor and is optional. Upon being asked for a title, the instructor may respond:

IEN=n

//NO

(title)

The first of these responses directs the generator to place in the object program the coding needed to retrieve the title that corresponds to the IEN he has selected. This will be the title used with the ESDP standard report form. For example, he may insert IEN=1.2 where 1.2 is the INPUT DATA DESCRIPTION in the standard report. In that case, the object program will contain a call to a subroutine that retrieves the stated title at object time. If no title is to be used, the instructor replies //NO. If, when the instructor is asked for a title, he gives any other response, that response is used in the object program. He might, in the earlier example, have used INPUT, preferring the shorter version of the standard title. In this case, the actual text is compiled into the object program.

c. Text

Essentially the same options are available to the instructor when text is called for by the generator program. In this case, if he replies IEN=n, the programmer-supplied text associated with n is implicated, not the title of n. Again, a call to a subroutine that will retrieve the text at object time is inserted into the object program, rather than the actual text.

d. Question

Questions will not have been stored by programmers before the instruction course is composed, so the instructor will always compose his own question, if he wants to use one. He may respond, then, with the text of the question or with //NO which means no question is to be asked.

e. Response Analysis

This is the most complex of the question components. First, the generator ascertains what form of question is to be used: multiple choice, etc. Then, a different set of interrogations may follow depending on what form of question is used. As an example, assume a true/false form is to be used. Then the generator automatically starts the response analysis component with the string:

```
IF RES = 'T' THEN DO;
```

where RES is the data item which contains the student's answer to the question.

For each test of an anticipated answer (such as T for TRUE) there is an associated THEN DO...END clause and an optional ELSE DO...END clause. The combined clauses are referred to as an

answer set: the set of code concerned with analyzing one anticipated answer. There is an answer set for each anticipated answer and one for unrecognizable answers.

The instructor is asked if TRUE is a correct answer to the question. If so, the next string generated is

```
GRADE(LABEL) = 1
```

where GRADE is an array of one-character, decimal items and LABEL is the subscript on the label array. If the answer were deemed incorrect, GRADE would be set to -1.

When the author selects multiple choice as a question form, the array of possible answers is elicited from him, to assist him in compiling the question. As soon as he selects the multiple choice mode, he is shown a display similar to

```
A.ENTER VALUE
```

The author then enters the first of the answers he wants his student to consider. The process is cumulative. As soon as the author gives a reply, that reply is displayed and a new line is created asking for the next possible answer. The sequence is terminated by //END and might take this form:

```
A. ENTER VALUE  _ABRAHAM LINCOLN

A. ABRAHAM LINCOLN
B. ENTER VALUE _GEORGE WASHINGTON

A. ABRAHAM LINCOLN
B. GEORGE WASHINGTON
C. ENTER VALUE  _//END
```

Now the instructor is asked about any additional processing he may want done by the object program at the point where the student has responded with a TRUE. His options are:

Execute an assignment statement

Type out a message

Count (i.e., add 1 to a designated data item)

Sum (i.e., add RES to some designated data item)

Store (i.e., store the contents of RES in a standard location, indexed by LABEL)

The count and sum options are intended for arithmetic processes other than keeping track of the student's grade. Using them does not require that the instructor enter a full PL/I statement, only that he designate which item is to be incremented.

Similarly, indication that a message is to be typed out requires only that the instructor enter the message, not that he enter a PL/I statement that would type out the message.

The only option that requires the instructor to enter actual programming statements is the assign option. This was included for those cases not covered by the more simple operations just described. If the assign option is used, the instructor enters a PL/I statement which is checked for validity by the generator. He must assume the responsibility for its correct application, but the generator assures him that only legal PL/I commands are put into the object program. While this feature has many uses, it should be apparent that fairly elaborate instructional programs can be composed without using it. The statement validity-checking subroutine is not yet implemented. At present, any character string entered in response to the question is accepted and assumed to be a PL/I statement.

The final step in each clause of an answer set is to find where the instructor wants to branch if the student has given the answer used in the test. (One restriction being imposed on the initial version of this program is that there may not be an IF following a THEN or ELSE. Another is that, in a multiple choice question, no information other than the response may be used to decide on branching. The capability to use other information will be inserted later, when item and text responses are permitted, for the logic involved follows immediately from the logic of processing those replies.) The instructor may select the question to which to branch by specifying any of:

> Back to a previous label (the label given must be on a list of generated labels)
>
> Forward to a new label (the label given may not have been generated yet)
>
> Forward to the next sequential question (with this choice it is not necessary to give the actual label)

Then there are several options for branching to other points within the current question. These are:

> Back to the text, so the student gets the entire tutorial section over again
>
> Back to the question, so the question is repeated but not the tutorial text

Back to response analysis, so a new answer is elicited and all analysis performed on it, but none of the foregoing text or question material is repeated

When branching to a different question, entry can only be made at the "top" of that question. The author cannot skip the heading.

Lists are maintained of all generated labels and all future labels (those indicated as forward branch labels at a branch point). Whenever the instructor goes on to a new question, he will be given a list showing from where else he branched to the same label. If he finds he has made a mistake and did not intend to go to the same label from two different points, he may change his most recent decision.

Branching is not actually done from the IF statement to the designated label. The IF statement will contain an assignment statement that stores the label to which branching outside the boundary of the UOI is to be done, then actual branching is to the ending component of the question.

As many IF statements, or answer sets, are generated as there are possible answers to check. Up to ten are allowed in a multiple choice. When full constructed item response analysis is implemented, there will be no limit except that imposed by memory considerations.

After all the answer sets are generated, an unrecognizable response answer set is inserted. In its general form this routine counts the number of successive unrecognizable answers, and prints out a different message for each. After three consecutive unrecognizable answers the student is cut off from the course (or the instructor from the use of the generator). The instructor provides the messages for the generated course. For example, if a True/False question is used and the student replies YRUE this answer would be unrecognizable. A message to this effect would be typed out and then another message, specific to the question and supplied by the instructor. For example, the student might see:

ANSWER UNRECOGNIZABLE

YOU MUST ANSWER TRUE OR FALSE

The first of these is inserted into the object course automatically by the generator, the second is provided by the instructor. If the student cannot give a recognizable answer in three tries, he is assumed to have grossly misunderstood the instructions or to be playing games, a common, apparently irrestible, urge among CAI students. The instructor may compose his own unrecognizable answer routine.

f. Ending

The ending routine gives the instructor the opportunity to do processing that applies to all answers, whether or not they match any of the anticipated answers, resets various counters, and performs the actual branch to the selected next question. The reason for doing the branching in this way is to enable the question to be operated as a subroutine, if necessary, or to be part of a sequential program, depending upon the value of some stated item. That is, if the student is in a review mode, he may be routed to a given question, then back to the review control question, regardless of this answer. In CAINT courses, we use this technique also for interrogating a programmer only on selected questions when he is updating files.

4. Composition of the Program. An object instructional program consists of up to 999 questions of the form just described (the number 999 being completely arbitrary, but some limit is necessary). When all the questions have been compiled, the generator program must generate introductory and terminal material for the object course.

Introductory material consists of required labels and statements of PL/I, such as data declarations. These are generated from lists of data items generated in the main body of the course. The terminal part of the course consists of a standard grade-computing routine, one that gives not only a count of right and wrong answers, but also a list of the number of unrecognizable answers at each question, and could be extended to give grades within major sections of the course. The terminal section also contains subroutines, completely pre-written, that must be a part of any object program.

The final result of operation of the generator program is a complete, syntactically valid PL/I program, together with all subroutines and specification statements. There can be no guarantee that the course will execute successfully, because the instructor may have inserted invalid or meaningless statements, but the course should always compile.

5. Using the System. The Instruction Generator is a new concept which offers a significant improvement in the potential for using CAI and PI for on-the-job training. Certainly, more time is needed to complete the generator and to test it, both as a program to be debugged and as an educational tool.

V

BIBLIOGRAPHY

[1] Hughes, J. L., Programmed Instruction for Schools and Industry. Science Research Associates, Chicago, 1962, p. 63.

[2] Crowder, Norman A., "Automatic Tutoring by Intrinsic Programming," in Teaching Machines and Programmed Learning: A Source Book, A. A. Lunsdaine and R. Glaser, Eds.,National Education Association, Washington, D.C. 1960, pp. 286-298.

[3] Lysaught, Jerome P. and Clarence M. Williams, A Guide to Programmed Instruction, John Wiley & Sons, Inc., New York, 1963 (a general survey of the subject).

[4] Skinner, B. F., "The Science of Learning and the Art of Teaching," in Teaching Machines, p. 100.

Security Classification

DOCUMENT CONTROL DATA - R & D

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

1. ORIGINATING ACTIVITY *(Corporate author)*
Center for Exploratory Studies
International Business Machines Corporation
Rockville, Maryland 20850

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
N/A

3. REPORT TITLE
EVOLUTIONARY SYSTEM FOR DATA PROCESSING
THE CAINT EXECUTIVE LANGUAGE AND INSTRUCTION GENERATOR

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
None

5. AUTHOR(S) *(First name, middle initial, last name)*
Charles T. Meadow
Douglas W. Waugh
Gerald F. Conklin
Forrest E. Miller

6. REPORT DATE
January 1968

7a. TOTAL NO. OF PAGES
42

7b. NO. OF REFS

8a. CONTRACT OR GRANT NO.
F19628-67-C-0254

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
ESD-TR-68-143, Vol. III

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Command Systems Division, Electronic Systems Division, Air Force Systems Command, USAF, L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

ESDP is a proposed system whose purpose is to acquire, store, retrieve, publish and disseminate all documentation, exclusive of graphics, concerned with a large computer programming activity. Documentation is deemed to consist, not only of final or formally published after-the-fact reports, but of working files, design and change notices, informal drafts, management reports--in fact, the entire recordable rationale underlying a programming system. Maximum attention has been concentrated on the means of acquiring and organizing documentation. Two major, complementary approaches are proposed. The first is called Program Analysis and is a process of extracting documentation directly from completed programs. The second is called Computer Assisted Interrogation and is a process of eliciting information directly from the programmer, through on-line communication terminals. The former provides canonical data about the program's structure. The latter provides explanatory material about all aspects of the program, and in the absence of canonical data, may provide tentative structural information as well. The conclusion of the study group is that ESDP is a feasible concept with present-day technology and that it will materially benefit using organizations in the production of programs and in guiding their evolution as requirements change. Its value will be greater for larger organizations, whose internal communications difficulties tend to cause truly gigantic inefficiencies. Its implementation as a support system for such projects would require a significant quantum of investment in order to produce these benefits and is predicated on the use of a computer system dedicated solely to the use of ESDP.

DD FORM 1 NOV 65 1473

Unclassified
Security Classification