AD/A-000 102

A COMPUTER-BASED SYSTEM FOR STUDIES IN LEARNING

Donald R. Gentner, et al

California University

Prepared for:

Office of Naval Research
Advanced Research Projects Agency

September 1974

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

AD/A-000 102

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Computer-Based System for Studies in Learning		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Donald R. Gentner, Mark R. Wallen, and Patricia L. Miller		8. CONTRACT OR GRANT NUMBER(s) N00014-69-A-0200-6045
9. PERFORMING ORGANIZATION NAME AND ADDRESS Prof. Donald A. Norman, Center for Human Information Processing, University of California, San Diego, La Jolla, California 92037		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 154-360
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel and Training Research Programs N00014, Office of Naval Research, Arlington, Virginia 22217		12. REPORT DATE September, 1974
		13. NUMBER OF PAGES 28
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Education, learning, computer-aided instruction, computer-based instruction, computer languages, automated tutor, model of the student, teaching.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes a computer-based system, called the FLOW system, used in experimental studies of human learning. The student learns a simple computer language from printed instructions and can run his programs interactively on the FLOW system. An automated tutor simulates a human tutor who watches over the student and gives help when the student has difficulties. The system also records detailed protocols of the interactions among the student, the computer, the automated tutor, and the human tutor for later analysis.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

A COMPUTER-BASED SYSTEM FOR STUDIES IN LEARNING

DONALD R. GENTNER

MARK R. WALLEN

PATRICIA L. MILLER

Contract Number: N00014-69-A-0200-6045

ARPA Order Number: 2284

Task Authority Identification Number: NR 154-360

Program Code Number: 3D20

Effective Date of Contract: 1 February, 1973

Contract Expiration Date: 31 January, 1975

Amount of Contract: $221,012

Organization: University of California, San Diego

Principal Investigator: Donald A. Norman

[Phone (714) 452-2947]

Scientific Officer: Dr. Marshall Farr, Director
Personnel and Training Research Programs
Office of Naval Research

This research was supported by the Advanced Research Projects Agency and the Office of Naval Research of the Department of Defense and was monitored by ONR under Contract No. N00014-69-A-0200-6045.

A COMPUTER-BASED SYSTEM FOR STUDIES IN LEARNING

by Donald R. Gentner, Mark R. Wallen, and Patricia L. Miller

University of California, San Diego

La Jolla, California 92037

INTRODUCTION

Our studies of learning have several major foci. First, how is information stored in the human memory system: how is new information integrated with the existing knowledge? Second, what are the mechanisms used by the learner to select information from the learning environment and integrate it with his existing knowledge structure? Third, how does the tutor model the knowledge state of the student and use that model to guide the tutorial dialog?

We have recently developed the FLOW system, a multipurpose facility for studies of the learning process. Some of the work with the FLOW system has been described previously (Norman, Gentner & Stevens, 1974; Norman, Gentner & Stevens, in press). This report will be concerned primarily with the development of an automated tutor which constructs a simple model of the student and uses that model to assist the student when needed.

Overview of the system

FLOW is a simple, interactive computer language whose commands are designed for string manipulation. Although FLOW is not intended for practical computer applications, it shares many important properties with common computer languages and is particularly suitable for our studies in learning. The section on the FLOW language describes the language in detail.

The FLOW system is based around a minicomputer. A cathode ray tube (CRT) display terminal for the student is attached to the minicomputer. The experimenter, who is normally in another room, can monitor the student's terminal on a television display. The minicomputer is connected to a large computer in which the automated tutor program resides. The experimenter has a terminal connected to the large computer which he can use to interact with the automated tutor program and, through that program, with the student. There are also facilities for recording complete protocols of the experimental session. The equipment used in the FLOW system is described in detail in the section on Hardware.

Over the past few years, a number of different approaches have been taken towards the development of computer-based instructional systems. These systems can be generally classed as one of three types. The first and most common type is based on the sequence: present text, test, and branch. The student is given a section of text material and then tested on that material,usually with a multiple choice test. Depending on the result of the test, the student can either be given the next text in sequence, or be sent back to a previous section, or be side-tracked to a remedial section. The second type of system is often called a generative system. The SCHOLAR system originally developed by Carbonell (1970) is a fine example of a generative system. Here the information (texts, questions and answers) are not explicitly stored, but instead there is a database in which information about the subject matter to be taught is stored as a semantic network. Generative routines in SCHOLAR can access the database to prepare text or questions for the student or to answer questions from the student. A third type of computer-based instructional system is the simulation system. In this type the computer is programmed to simulate some domain of interest, such as a regulated power supply (Brown, Burton & Bell, 1974) The student has a number of operations he can perform on the simulation and learns by seeing the results of these operations.

Actually, Brown's system can also carry on simple dialogs with the student. Our FLOW system takes a similar approach, giving the student an interactive domain to experiment with (an implementation of the FLOW language) and tutorial assistance. (For examples of related systems see Barr, Beard & Atkinson, 1974; Koffman, Blount, Gilkey, Perry & Wei, 1973; Goldberg, 1973.) The automated tutor in the FLOW system, however, differs considerably from previous systems. In some preliminary experiments, the student learned FLOW primarily from written instructions, entered his programs into the computer and modified his programs based on the results. Meanwhile a human tutor was watching over the shoulder of the student and would answer questions or interrupt if the student appeared to be having difficulty. The automated tutor is meant to duplicate the function of that human tutor. The student still learns primarily from written instructions, but the autotutor tries to follow his progress and keep track of where he is in the instructions by monitoring what he types into the computer. The student need not follow the written instructions exactly, but may move back or skip ahead. The autotutor will try to follow and be ready to give appropriate help if the student gets in trouble.

THE FLOW LANGUAGE

Origins

The FLOW language was originally developed by Professor Jeffrey Raskin of the Visual Arts Department at UCSD. He wanted to teach computer programming to humanities students who often either had little experience with, or actively disliked mathematics and computers. FLOW was conceived as an introductory computer language that would be non-threatening and easy to understand while still possessing the basic functions and constructions of typical computer languages. After first learning FLOW, the student would then go on to a standard language such as BASIC (Raskin, in press).

FLOW

Table 1 shows the commands and statements in the implementation of FLOW used in these experiments. (After the experiments described in this report, we renamed most of the FLOW commands and statements to make them more mnemonic. See the section on Subsequent Developments.)

In our implementation of FLOW, the student's terminal is connected to the minicomputer via a full-duplex link. This means that when the student presses a key on his terminal, the character is first sent to the minicomputer, examined by the computer and then echoed back for display on the terminal. This arrangement allows two interesting features. The first feature is called "typing amplification" by Raskin. As soon as the minicomputer can unambiguously recognize a command or statement from the characters input by the student it supplies the remaining letters of that command. Since most FLOW commands begin with different letters, it is usually necessary to type in only the first letter of a statement. With the commands and statements listed in Table 1, only the underlined letters need to be typed in, the computer will supply the rest. Second, since the computer can examine each character before displaying it on the terminal, syntactically incorrect characters can be rejected. Together these two features virtually eliminate typing and syntactic errors, two major sources of frustration for beginning programmers. To illustrate how they work, suppose a student is entering his program. After completing a line, the computer does a carriage return and line feed, provides a new line number and then waits for a student input. At this point the minicomputer will accept any letter which begins a FLOW command or statement or a number (indicating that the student wants to change the line number). If the student presses the "P" key the word "PRINT" is displayed on the terminal and the minicomputer waits for a legal completion of a PRINT statement. (The legal characters at this point are I, ', and R). If the student then presses "J", the minicomputer will momentarily display a "J" with an audible tone then backspace, erase the "J" and wait for a legal character. An "I" will be accepted as a legal character; the minicomputer displays "IT" does a carriage return, linefeed, displays a new line number and waits for the next statement. It is not a recommended programming practice, but as a graphic demonstration of these two features, it is possible to bang away at the keyboard, and always produce syntactically correct, executable FLOW programs, although, of course, there is no guarantee of what they will do.

Advantages of FLOW for Learning Studies

FLOW has a number of advantages as a subject matter for use in studies of learning. Along with other computer languages, FLOW involves both conceptual and procedural knowledge. It is problem oriented. There are a large variety of problems that can be posed to exercise the student's developing knowledge. The language itself and the tasks form a small, concise, and well defined body of knowledge. The statement of a problem and what constitutes an acceptable solution are normally fairly clear, although there may be some debate about the quality of a particular solution. The students in most of our studies were university undergraduates, and it is relatively easy to find subjects from that group with little or no experience with computer languages.

FLOW also has some unique advantages as a subject matter. With typing amplification and the rejection of syntactically incorrect characters, two major sources of uninteresting errors are eliminated. The errors we are left with are concerned with the meaning of individual statements and how groups of statements interact. Since FLOW is quite a simple language, a lot of interesting learning takes place within the first hour. The subject matter is complex enough, however, to require up to 10 hours to master. This forms a convenient time span for experimental studies. Subjects usually enjoy learning FLOW with its interactive nature and low level of frustration, and are highly motivated to learn.

The last advantage of FLOW derives from the fact that it is primarily a computer mediated task. This leads naturally to the possibility for computer interaction during the instructional sequence and to ease of collection of experimental data. Also since the task is primarily verbal, it is easy to collect detailed protocols of the tutorial interaction and attempts at problem solving for later analysis.

HARDWARE

Although this report is concerned mainly with the automated tutor, the FLOW system is designed for a number of different types of experimental studies. Figure 1 is a schematic diagram of our equipment. The equipment (except for the B6700) is located in two adjoining experimental booths.

The system is based around a Microdata 810 minicomputer with 24K bytes of memory (the applications described here require only 8K bytes). The student works at a Scientific Measuring Systems 1440 CRT terminal connected to the minicomputer via a full-duplex 1200 baud (120 characters per second) link. A video output from the CRT terminal is connected to a video monitor in the experimenter's booth duplicating the display on the student's terminal. The signal going from the minicomputer to the student's terminal which contains all of the information to be displayed on the terminal including vertical and horizontal spacing also goes to a recording unit. This consists of a modem converting the digital signal produced by the minicomputer into a sequence of tones which are then recorded on an audio tape recorder. The tape recorder can record up to four tracks of information such as verbal comments from the student and tutor in addition to the terminal display. Later when the tape recording is played back through the modem, the tones are converted back into digital signals which faithfully duplicate the experimental session on the CRT terminal.

The automated tutor part of our system is contained in programs residing on the UCSD's Burroughs 6700 computer, which is connected to our minicomputer via two half-duplex 4800 baud lines. (The two half-duplex lines can be replaced with a single full-duplex connection when the system will support full duplex.) The B6700 computer is also connected to a Tektronix 4010 display terminal in the experimenter's booth which is used to program the B6700 and to communicate between experimenter and the automated tutor during tutorial sessions. In additicn the B6700 prepares a protocol of the tutorial session for later analysis. The experimenter's booth also contains a keyboard and Teletype terminal which are used to communicate with the minicomputer.

The minicomputer controls most of the flow of information. It monitors the student's keyboard, interprets, and runs his programs, and writes on the student's display screen. Input from the student is summarized and sent to the automated tutor on the B6700 and all messages from the autotutor are relayed through the minicomputer. The major information link outside the minicomputer is that the experimenter can interact directly with the autotutor.

THE AUTOMATED TUTOR

The Printed Instructions

We decided to base the teaching of FLOW on a set of printed instructions primarily because of ease of preparation, flexibility, and convenience of use for the student. Few new educational technologies can rival the printed page which is self-paced, portable, easily searched and inexpensive. Figure 2 shows a typical portion of the instructions.

The instructions are divided into seven units, each of which involves three types of tasks: reading, entering practice programs, and solving problems. These three types of tasks can be seen in Figure 2. The first part of the figure is explanatory text. Then there is a short program using the new statement which the student can enter and run. Finally the student is given a programming problem to solve. All the units have this structure; there are one or more sections of text and practice programs, and then the student is given a problem to solve at the end of the unit. We ask the student to not go on to the next section until he is told to. In fact the autotutor will send him on as soon as he solves the problem. The student can go back in the instructions whenever he likes but we thought he should not move on to the next unit until he had mastered the current unit. We were also afraid the autotutor might have difficulty if it thought the student was working on Problem 4 when he was actually working on Problem 5.

New commands and statements are introduced in each section. With typing amplification naive students often accidentally enter statements which they know nothing about, leading to general confusion. To avoid this problem, the minicomputer initially does not respond to any key from the student's keyboard. As the student moves through the instructions,keys are activated at the time that the corresponding statements are introduced. While the student is working on Unit 1, for instance, only the H,P,',R,L,N and E keys are activated (corresponding to HELP, PRINT 'string', RUN, LIST and NEW). When the student successfully completes the problem at the end of unit 1, the **autotutor** gives him a congratulatory message, tells him to go on to Unit 2, and activates the keys needed for that unit.

Following the Student

During the development of the autotutor, all its functions were originally carried out by a human tutor. Parts of the tutorial function were then taken over by the computer until it was completely automated. The human tutor typically forms a rather simple model of the student containing the following information:
1) where the student is in the instructions, and 2) his current program. The human tutor refers to the instructions and general knowledge to generate: 3) the student's next expected input, and 4) the maximum time it should take him to enter the input. If the student entered the expected input, the human tutor would advance the student model to the next point in the instructions. If the student entered an unexpected input, the tutor would use the instructions and general world knowledge to try and deduce why the student had done that, and where he now was in the instructions. Rather than have the autotutor generate maximum pause

lengths and the meaning of unexpected inputs from the instructions as a human tutor does, we decided that that would be too difficult a task and entered the maximum pause length and a list of inputs explicitly for each state.

One of the major tasks of the autotutor is to follow the student. The student normally proceeds sequentially through the instructions. But from time to time he may make errors and decide to repeat things or even go back to an earlier part of the instructions. A confident student may skip over some of the practice programs and start immediately on the problem. Through all this, the autotutor monitors what the student is typing into his terminal and tries to keep track of where he is in the instructions.

The instructional sequence is divided into 136 states. During the practice parts of the instructions, a state usually corresponds to a FLOW statement or command, although in the first unit, states may correspond to individual characters and partial commands. During the problem part of the instructions, each problem corresponds to essentially a single state. Each state has associated with it a list of information which the autotutor uses to follow the student. These states and their associated lists are all that the autotutor knows about the instructions. Because the autotutor employs two distinct strategies for the practice states and the problem states, we will discuss each of these separately.

Practice

The autotutor gives three types of messages during the practice part of a unit: 1) standard messages in response to a request for help, or if the student pauses too long, based on the next expected input

(Try typing a_______); 2) messages in response to a serious error, (usually a NEW out of sequence) which tell what should have been typed and send the student back to a previous section to try again (You should have typed _______; please go back to Section _____); 3) special messages peculiar to a given state or class of states which clarify usage of certain keys or commands or explain procedures (e.g., If during the first unit the student types the letter "O" when he should have typed the number "0", he gets the message, "Zero (0) is located to the right of 9. Don't use the letter O for zero.")

To illustrate how the autotutor works when the student is in the practice part of the instructions, let's look in detail at one section of the instructions in Figure 2. In this unit we introduce the PRINT RETURN statement. In the practice part we give a brief explaration of the statement and then a practice program using the new statement which the student is expected to enter and run.

When the student successfully completes the previous problem (Problem 2) the autotutor's model of the student enters State 276. (The states are not all numbered consecutively.) When the student then enters a NEW command, the minicomputer erases the screen and gives the student a line number (010) for his new program. Meanwhile the autotutor advances the student model to State 277. We will examine State 277 in detail.

Associated with State 277 are four possible student inputs and a maximum pause length. The autotutor thinks that the student is about to type in the practice program and waits for a student input. The five inputs which it is looking for in particular are:

PRINT 'HELLO'	This is what the autotutor expected and it simply advances the student model to the next state (State 278).
PRINT 'BASKET'	This is the first statement for the solution of Problem 2, so the autotutor assumes that the student has skipped the practice section and advances the student model to the beginning of Problem 2 (State 282).
PRINT 'anything else'	The autotutor assumes that it just has a creative student who didn't like the word HELLO and advances him to State 278.
NEW	The autotutor assumes that the student had made some errors and decided to begin again. The student model is kept in State 277, and to keep some control over the chaos, the student is asked to start at Section 13.
HELP	The autotutor sends the message: TRY TYPING A PRINT 'HELLO'. The student model remains in State 277.

Any other input is assumed to be an error which the student will notice and correct. The student model is left in State 277.

The maximum pause length for State 277 is 30 seconds. Failure to press any key for 30 seconds is equivalent to typing HELP. The student gets the message "TRY TYPING A PRINT 'HELLO'" and the student model remains in State 277. (The maximum pause lengths, which vary from 30 seconds to 5 minutes are entered explicitly by the experimenter.)

Problems

When the student finishes the practice part of the instructions, the student model is advanced to the state corresponding to the next problem. A problem essentially corresponds to a single state; running a correct solution to the problem advances the student to the next state. Associated with each problem state, the autotutor has a program which is a correct solution of the problem. If the program requires input (in FLOW, that would be in the form of a TEXT statement), the autotutor will also have several test inputs. As the student types in his program, the minicomputer sends the statements to the autotutor.

If the student runs his program, asks for help or pauses too long, the autotutor runs the student's program along with the correct program, substituting in its test inputs if necessary. If the two programs generate essentially the same output, the student's program is judged correct, and the student is given a congratulatory message and sent on to the next unit. The autotutor is somewhat lenient about certain discrepancies in the student's output such as extra spaces. Within the rather small scope of FLOW and the programming problems we use, our procedure of comparing the outputs of a correct program and the student's program works quite well. If the outputs are not essentially the same, the autotutor does a line by line comparison of the programs and informs the student of the first discrepancy it finds between the student's program and the correct program.

The program is viewed topologically as groups of non-control statements bounded by control statements. (Control statements alter the linear flow of command of execution; they are STOP, JUMP TO nnn, and IF IT IS 'c' JUMP TO nnn.) The non-control parts are examined statement by statement as previously described. When a control statement is encountered, the branch is taken to the new portion of the program and this is examined. This is a recursive process, and if this new portion passes inspection, then the statement following the initial control statement is examined. Thus the whole executable program will be canvassed for errors.

Example of a Tutorial Interaction

To illustrate the action of the autotutor, we will present a portion of a tutorial interaction along with the autotutor's interpretation of the situation in terms of its modification of the student model. Although this dialog is taken from actual tutorials, we have blended together several tutorials with different students to be able to show a variety of student errors and how the autotutor handles them. As we start this excerpt, the student has just correctly solved Problem 2.

Autotutor

Student Model State	Next Expected Input	Student Input	Messages from Autotutor	Comments
276	NEW	NEW		Removes the old program
277	PRINT 'HELLO'	HELP	TRY TYPING A PRINT 'HELLO'	
277	PRINT'HELLO'	PRINT 'HELLOO'		The "OO" is acceptable.
278	PRINT RETURN	NEW	YOU SHOULD HAVE TYPED >>PRINT RETURN<<. PLEASE GO BACK TO SECTION 13	The student didn't like the "OO" and decided to start over. This message from the autotutor was probably a bit annoying.
277	PRINT 'HELLO'	PRINT 'HELLO'		
278	PRINT RETURN	PRINT RETURN		
279	PRINT 'GOODBYE'	PRINT 'GOODBYE'		
280	RUN	RUN		
281	NEW	NEW		The student starts on Problem 2
282	any input	HELP	MAKE THE COMPUTER PRINT AN ENDLESS VERTICAL LINE OF BASKETS, ONE PER LINE.	This message is given only if the student asks for help immediately after entering a problem state.
283	Solution to Problem 3	010 PRINT 'BASKET'		
283	" "	Student pauses for more than 60 sec.	PLEASE REREAD SECTION 9	Pausing too long is equivalent to asking for help. Section 9 is on the JUMP statement.

Autotutor

Student Model State	Next Expected Input	Student Input	Messages from Autotutor	Comments
283	Solution to Problem 3	020 JUMP TO 010		
283	" "	RUN		An infinite loop.
283	" "	HELP	YOU NEED A >>PRINT RETURN<< STATEMENT.	The autotutor now compares the student's program with the correct program.
283	" "	030 PRINT RETURN		It's in the wrong place.
283	" "	RUN		
283	" "	HELP	YOUR >>PRINT RETURN<< STATEMENT AT 030 IS IN THE WRONG PLACE.	
283	" "	NEW		
283	" "	010 PRINT 'BASKET'		
283	" "	020 PRINT RETURN		So far, so good.
283	" "	030 JUMP TO 020		Oops
283	" "	RUN		
283	" "	HELP	YOUR >>JUMP TO 020<< SHOULD JUMP TO A >>PRINT 'BASKET'<< STATEMENT.	The autotutor found that although the student's program was similar to the correct program, the JUMP statements led to different statements.
283	" "	030 JUMP TO 010		Student substitutes correct statement.
283	" "	RUN	EXCELLENT WORK! YOU MAY NOW GO ON TO PAGE 8, PROBLEM 5.	The autotutor compares outputs of the student's program and the correct program on every RUN. This time they matched.

Summary of the Automated Tutor

As the student works from written instructions, the autotutor monitors the student's keyboard inputs and tries to follow the student's progress through a series of states corresponding to the written instructions. The states corresponding to the practice parts of the instructions have an expected pause length and a list of possible inputs associated with them. Corresponding to each possible input is a destination state and possibly a message. In a given state, if the student types in one of the inputs on the list, the autotutor prints out the message if any and advances the model of the student to the destination state. If the student types in an input not on the list, the student model remains in the same state. Except for these messages, which are rare, the only time that the student is aware of the presence of the autotutor is when he asks for help or pauses for a long time and exceeds the expected pause length. Then the autotutor prompts the student with the next expected input.

Somewhat surprisingly it has been our experience that the autotutor usually makes fewer errors in following the student through the practice parts than a human tutor. Analysis of protocols from early sessions with human tutors showed that the human tutors often were quite mistaken in their judgments of where the student was in the instructions. The main problem seemed to be that crucial key presses were missed due to lack of attention. Of course, eternal vigilance is one of the supreme virtues of a computer, and this seems to give it the edge in this task.

The operation of the autotutor while the student is working on a problem is much different. Here the autotutor only has an example of a correct solution to the programming problem. With the exception of two messages which are sometimes given early in the problem, all of the messages from the autotutor are generated algorithmically based on a comparison of the student's program and the correct program. In addition to assisting the student with an incorrect program, the autotutor also runs the student's program whenever the student runs it, and if the program seems correct, the autotutor congratulates the student and asks him to start on the next unit.

SUBSEQUENT DEVELOPMENTS

Command Names

Students learning FLOW often found some of the names of FLOW statements and commands obscure or confusing. In an effort to clarify their functions, we have renamed many of the instructions. Table 2 lists the old instructions and the corresponding new instruction.

We have also altered the display generated by the WALK command to make it clearer what is happening: eight lines of the student's program are displayed on the upper portion of the screen along with a pointer indicating the statement currently being executed.

Conceptual FLOW

One of our original interests was to teach students the FLOW commands and statements and study the spontaneous development of programming concepts. Thus the written instructions and automated tutor described in this report are oriented around the statements in the FLOW language. Concepts such as control transfer and loops are not explicitly treated. Some tutorials we have conducted indicate that instruction based on programming concepts rather than the specific statements would be more fruitful, especially for students with no previous experience with computers or programming. To produce a FLOW instructional system similar to the one discussed here but with a conceptual emphasis would involve rewriting the instructions and modifying the autotutor to analyze student programs in terms of the relevant concepts. An interest of ours, in another part of the FLOW study, is the manner in which students form their developing knowledge into schemata. These schemata usually seem to be concerned with concepts such as loops. A version of instruction in FLOW oriented towards the programming concepts would thus also connect in more directly with our studies of schemata and how they are used in learning.

REFERENCES

Barr, A., Beard, M., and Atkinson, R.C. A rationale and description of the BASIC instructional program. (Technical Report No. 228) Stanford, California: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1974.

Brown, S. , Burton, R.R., and Bell, A.G. Sophie: A sophisticated instructional environment for teaching electronic troubleshooting. (BBN Report No. 2790) Boston: Bolt Beranek and Newman, Inc., 1974.

Carbonell, J.R. AI in CAI: An artificial-intelligence approach to computer-assisted instruction. IEEE Transactions on Man-Machine Systems, 1970, MMS-11, 190-202.

Goldberg, A. Computer-assisted instruction: The application of theorem-proving to adaptive response analysis. (Technical Report No. 203) Stanford, California: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

Koffman, E.B., Blount, S.E., Golkey Gilkey, T., Perry, J., Wei, M. An intelligent CAI monitor and generatave tutor. (Interim Report) Storrs, Connecticut: Computer Science Program, 1973.

Norman, D.A., Centner, D.R., and Stevens, A.L. Learning and Teaching. (Technical Report) La Jolla, California: Center for Human Information Processing, University of California, 1974.

Norman, D.A., Gentner, D.R., Stevens, A. Teaching and learning as a communication process. In D. Klahr (Ed.), Cognition and Instruction: Proceedings of the Fifth Annual Carnegie Symposium. In press.

Raskin, J. Flow, a language for newcomers to computers, Computers and the Humanities. In press.

TABLE 1

FLOW INSTRUCTIONS

Commands	
HELP	Requests help from tutor or autotutor.
NEW	Erases current program.
RUN	Runs current program.
WALK	Steps through the current program. One line of the program is executed each time the space bar is pressed. The program statements are displayed as they are executed on the upper part of the screen, variables are displayed in the middle of the screen, and any resulting output is displayed in the lower part of the screen. This command is a very useful debugging aid.
LIST	Displays the current program.
Program Statements	
PRINT 'character string'	Displays the character string.
PRINT RETURN	Does a carriage return and line feed.
PRINT IT	Displays the variable IT.
JUMP TO nnn	Statement number nnn is executed next. (If there is no statement with line number nnn, the statement with the next highest line number is executed. If all the line numbers are lower than nnn, the program stops.)
TEXT IS 'string'	Defines a character string to be used as data.
GET IT	Sets the variable IT to successive characters in the TEXT statement.
IF IT IS 'x' JUMP TO nnn	If the variable IT is the same as the character in quotes, the JUMP is performed.
COMMENT character string	No effect.
MAKE COUNTER ZERO	
ADD ONE TO COUNTER	
DECREASE COUNTER BY ONE	
PRINT COUNTER	Displays the variable COUNTER.
IF COUNTER IS ±dddd JUMP TO nnn	

TABLE 1 (cont'd.)

Notes: System Commands are executed immediately on entry. Program statements are used to write stored programs.

The COMMENT statement and the ones below it were not used in the studies described in this report.

The programmer types in only the underlined characters, the computer supplies the rest. (See the section on Typing Amplification.)

The programmer must type in the first two letters of the NEW command. This is designed to help prevent accidental erasure of programs.

The lower case letters in the statements correspond to slots to be filled in by the programmer. In particular, the phrases "character string" or "string" can be replaced by any string of characters and the phrases "dddd" and "nnn" are meant to be replaced by numbers of up to 4 and 3 digits respectively.

TABLE 2

NEW FLOW INSTRUCTIONS

Old Instruction	New Instruction
HELP	HELP
NEW	ERASE
RUN	RUN
WALK	WALK
LIST	LIST
COMMENT string	COMMENT string
PRINT 'string'	DISPLAY "string"
PRINT IT	DISPLAY VARIABLE
PRINT COUNTER	DISPLAY COUNTER
PRINT RETURN	BEGIN ON NEXT LINE
JUMP TO nnn	JUMP TO nnn
TEXT IS 'string'	TEXT IS "string"
GET IT	GET VARIABLE
IF IT IS 'x' JUMP TO nnn	IF VARIABLE IS "x" JUMP TO nnn
STOP	QUIT
MAKE COUNTER ZERO	MAKE COUNTER ZERO
ADD ONE TO COUNTER	ADD ONE TO COUNTER
DECREASE COUNTER BY ONE	SUBTRACT ONE FROM COUNTER
IF COUNTER IS ±dddd JUMP TO nnn	IF COUNTER IS ±ddd JUMP TO nnn

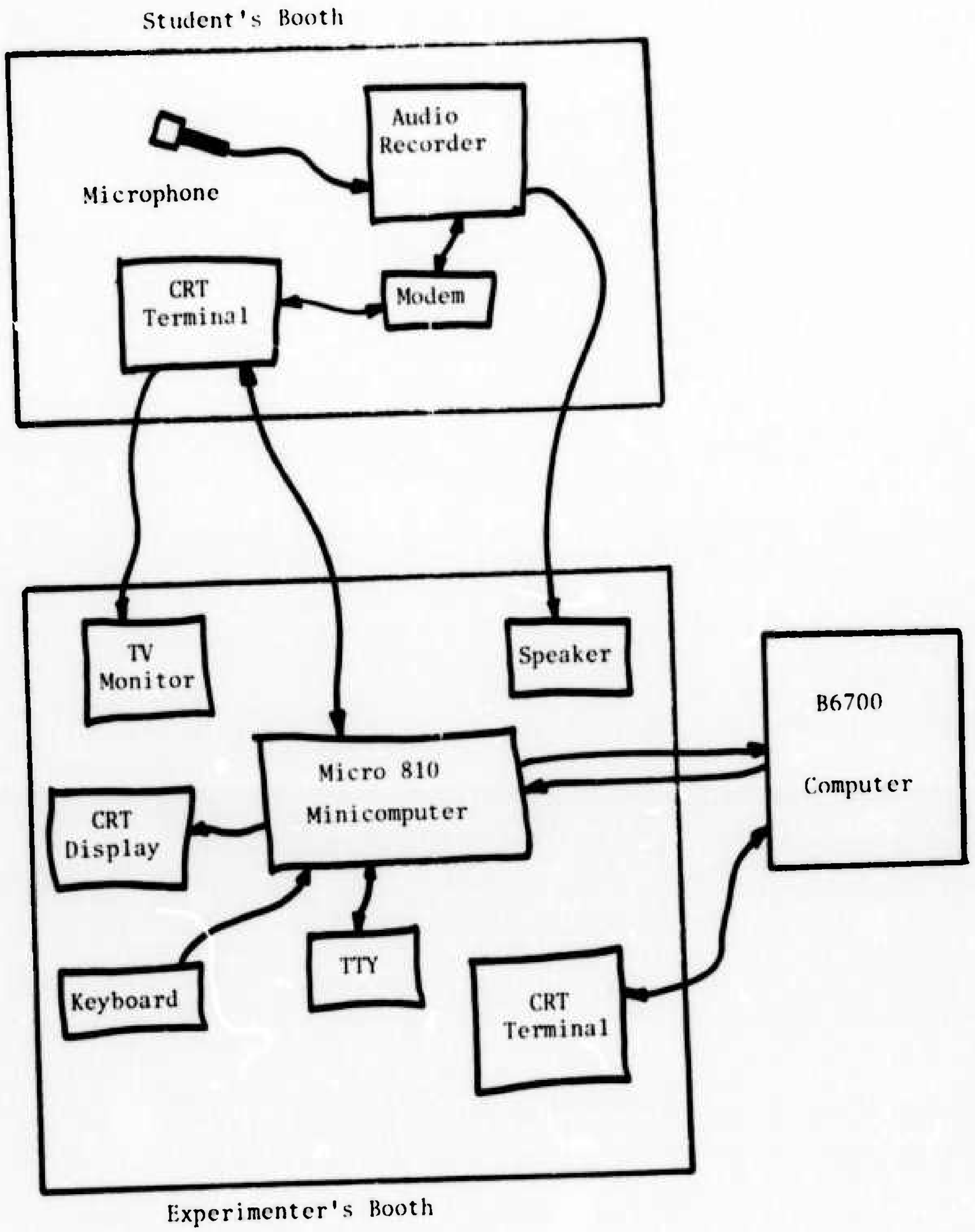

Figure 1

The FLOW Expe~imental Facility

PROBLEM 3:

<u>Section 12</u>

You are now going to learn how to make the computer type a vertical string of BASKETs, with each word on a separate line. That is, the result should look like this:

```
        BASKET
        BASKET
        BASKET
           :
           :
        BASKET
```

<u>Section 13</u>

You should find this an easy program to write. But you need one more instruction: you need to know how to make the computer print on a new line. We do that by printing a carriage return:

```
        PRINT RETURN
```

(Note that no quote marks are used for this command.)

Here's a program using <u>P</u>RINT <u>R</u>ETURN:

```
        010 PRINT 'HELLO'
        013 PRINT RETURN
        020 PRINT 'GOODBYE'
```

Type <u>NE</u>. Then type in this program and run it.

The computer will print:

```
RUN
HELLO
GOODBYE
HALT
030
```

<u>Section 14</u>

Now try Problem 3: Modify your program from Problem 2 (see Sec. 9) to make the computer print an endless vertical string of BASKETs, one on each line, using the procedures given on Page 3 for modifying your program.

After you've made your modifications, list your new program to see what it looks like. Then run the program to be sure it works.

Figure 2

A Portion of the Printed Instructions