DOCUMENT RESUME

ED 230 200 IR 010 723

AUTHOR Mayer, Richard E.
TITLE Diagnosis and Remediation of Computer Programming Skill for Creative Problem Solving. Volume 2: Summary of Research for Practitioners. Final Report.
INSTITUTION California Univ., Santa Barbara.
SPONS AGENCY National Inst. of Education (ED), Washington, DC. Teaching and Learning Program.
PUB DATE Oct 82
GRANT NIE-G-80-0118
NOTE 106p.; For related document, see IR 010 722.
PUB TYPE Reports - Descriptive (141)

EDRS PRICE MF01/PC05 Plus Postage.
DESCRIPTORS *Calculators; *Computer Literacy; *Computers; *Curriculum Development; Design Requirements; Individual Differences; *Learning Processes; Man Machine Systems; Models; *Programing; Programing Languages; Teaching Methods
IDENTIFIERS BASIC Programing Language; Chunking; Learning Strategies

ABSTRACT

This three-part volume provides a summary, for use by practitioners, of a project concerned with how novices learn to become creative educational computer users. The first chapter examines techniques for increasing the novice's understanding of computers and computer programming, and specifically analyzes the potential usefulness of five recommendations for the design of computer literacy curricula. For each recommendation, a statement of the issue, an example, relevant background, and a brief review of the relevant research literature are provided. In the second chapter, a framework is presented for describing users' knowledge of how a simple 4-function calculator operates, and data on individual differences among novices and experts in their conceptions of a calculator's internal operations for various sequences are summarized. The third chapter provides a summary of a study in which 30 undergraduates learned BASIC programming language through a self-paced mastery manual and were tested on their mental models for the execution of ea[illegible] of nine BASIC statements. A catalog is then presented of beginning programmer's conceptions of the internal functions of the computer for each of the BASIC statements based on test results. Each section includes tables and references. (LMM)

Final Report

Grant NIE-G-80-0118

DIAGNOSIS AND REMEDIATION OF COMPUTER PROGRAMMING SKILL FOR CREATIVE PROBLEM SOLVING

Volume 2

Summary of Research for Practitioners

October, 1982

Richard E. Mayer, Principal Investigator

The Program on Teaching and Learning, U. S. National Institute of Education, funded this project through Grant NIE-G-80-0118 to the University of California, Santa Barbara.

The purpose of this volume is to provide a summary of the project for use by practitioners. A more detailed description of the project methods and results is presented in Volume 1.

Table of Contents (Volume 2)

CHAPTER 1

LITERATURE REVIEW

Note

The material in this chapter has been published as the following article:

Mayer, R. E. Contributions of cognitive science and related research on learning to the design of computer literacy curricula. In R. J. Seidel, R. E. Anderson, & B. Hunter (Eds.), Computer Literacy: Issues and Directions for 1985. New York: Academic Press, 1982, 129-161.

This report was also published as a technical report:

Mayer, R. E. Contributions of cognitive science and related research on learning to the design of computer literacy curricula. Technical Report No. 81-1. Santa Barbara: University of California, 1981.

The goal of this paper is to examine techniques for increasing the novice's understanding of computers and computer programming. In particular, this paper examines the potential usefulness of five recommendations concerning the design of computer literacy curricula, as listed below.

(1) Provide the learner with a concrete model of the computer.

(2) Encourage the learner to actively restate the new technical information in his or her own words.

(3) Assess the learner's existing intuitions about computer operation and try to build on them, or modify them, as needed.

(4) Provide the learner with methods for chunking statements into a larger, single, meaningful unit.

(5) Provide the learner with methods for analyzing statements into smaller, meaningful parts.

For each recommendation, this paper will provide a clear statement of the issue, an example, relevant background, and a brief review of relevant research literature.

As can be seen, each recommendation is concerned with increasing the meaningfulness of learning new computer information by novices. For purposes of the present paper, meaningful learning is viewed as a process in which the learner connects new material with knowledge that already exists in memory (Bransford, 1979). Figure 1 provides a general framework for discussing the conditions of meaningful learning (see Mayer, 1975, 1979a). The figure shows that information enters the human cognitive system from the outside (e.g., through text or lectures, etc.), and must go through the following steps: (1) <u>Reception</u>. First, the learner must pay attention to the incoming information so that it reaches working memory, as indicated by arrow a. (2) <u>Availability</u>. Second, the learner must possess appropriate prerequisite concepts in long term memory to use in assimilating the new information, as indicated by point b. (3) Activation. Finally, the learner must actively use this prerequisite knowledge during learning so that the new material may be connected with it, as indicated by arrow c from long term memory to working memory. Thus, in the course of meaningful learning, the learner must come into contact with the new material (by bringing it into working memory), then must search long term memory for what Ausubel (1968) calls "appropriate anchoring ideas", and then must transfer those ideas to working memory so they can be combined with the new information in working memory. Each recommendation is aimed at insuring one or more of these conditions is met.

The traditional way of evaluating meaningful learning is to test whether learners can transfer what they have learned to new situations. For example, Wetheimer (1959) taught students how to find the area of a parallelogram using a rote method (i.e., memorizing a formula) or a meaningful method (i.e., involving the structure of the figure). Although both groups performed equally well on problems like those given during instruction, Wertheimer claimed that the meaningful learners were able to solve unusual problems requiring creative transfer. Thus, this paper will focus on transfer as a measure of meaningful learning of computer programming.

1. USE CONCRETE MODELS

Statement of the Problem

Novices tend to lack domain specific knowledge (Greeno, 1980; Simon, 1980; Spilich, Vesonder, Chiesi & Voss, 1979). Thus, one technique for improving the novice's understanding of new technical information is to provide them with a domain-specific framework that can be used for assimilating new information--i.e. by allowing for "availability" as indicated by point b in Figure 1. The present section focuses on the effects of concrete models on people's understanding of computers and computer programming.

Example

For example, in our own work on teaching a simple BASIC-like language to novices, we presented a model of the computer such as shown in Figure 2. The model provides concrete analogies for four functional units of the computer: (1) input is represented as a ticket window in which data is lined up waiting to be processed and is placed in the finished pile after being processed, (2) output is represented as a message note pad with one message written per line, (3) memory is represented as an erasable scoreboard in which there is natural destructive read-in but non-destructive read-out, and (4) executive control is represented as a recipe or shopping list with a pointer arrow to indicate the line being processed. This model may be presented to the learner either as a diagram, or as an actual board containing these useable parts.

Background

There is ample evidence that concrete models are widely used in mathematics instruction. For example, early work by Brownell & Moser (1949) indicated that children who learned subtraction algorithms with the aid of "bundles of sticks" were better able to transfer to new problems than children who were given the rules for subtraction in abstract form with plenty of "hands on" experience in executing the procedures. More recently, the important role of "manipulatives" such as coins, sticks, blocks, etc., has been documented by Weaver & Suydam (1972) and by Resnick & Ford (1980).

There is also some evidence that concrete models may enhance comprehension of text. For example, students' recall of an ambiguous passage was enhanced when a title or diagram or introductory sentence was given prior to reading but not when given after reading (Bransford & Johnson, 1972; Dooling & Lachman, 1971; Dooling & Mullet, 1973). Similarly, Ausubel (1963, 1960, 1968) has provided some evidence that expository learning may be enhanced by using an "advance organizer"--a short, expository introduction, presented prior to the text, containing no specific content from the text and providing a general framework for subsuming the information in the text. More recent reviews of the advance organizer literature reveal that advance organizers tend to have their strongest effects in situations where learners are unlikely to already possess useful prerequisite concepts--namely, for technical or unfamiliar material, for low ability subjects, and when the test involves transfer to new situations (Mayer, 1979a, 1979b).

Royer and his colleagues (Royer & Cable, 1975, 1976) have demonstrated that concrete models may serve as effective advance organizers in learning new scientific information. For example, the concrete analogy of electrical conduction as a chain of falling dominoes, influenced subsequent learning. Similarly, White & Mayer (1980) analyzed the concrete models used by physics textbooks. For example, Ohm's Law is described in terms of water flowing in pipes, a boy pushing a heavy load up an inclined street, or electron flow in a circuit. Recent results by Cook & Mayer (1980) show that when concrete analogies are embedded in a technical text, novices tend to perform best on recalling these familiar models and tend to recognize the information related to the models.

DuBoulay and his colleagues (DuBoulay & O'Shea, 1976; DuBoulay, O'Shea & Monk, 1980) have distinguished between two approaches to learning computer programming. In the <u>black box</u> approach, the operations of the computer are hidden to the learner so that the learner has no idea of what goes on inside the computer. In the <u>glass box</u> approach, the user is able to understand the changes that occur inside the computer for each statement. Although the description need not, indeed should not, be on a machine language level, DuBoulay et. al. (1980) suggest two properties for making the hidden operations of a computer language more clear to the novice: simplicity--there should be a "small number of parts that interact in ways that can be easily understood", and visibility--novices should be able to "view selected parts and processes of this notational machine in action". DuBoulay et. al. have implemented these suggestions in an instructional course in LOGO, since each statement is related to a concrete model called "the LOGO machine". However, there is yet no empirical test concerning the effects of the LOGO machine on learning.

Research of Concrete Models

Transfer. In order to provide some information concerning the role of models on learning computer programming, a series of studies was conducted (Mayer, 1975). In the studies, subjects were either given a concrete model of the computer (such as shown in Figure 1) or not; then, all subjects read a 10-page manual describing seven BASIC-like statements (see Table 1). Following reading, subjects took a test that consisted of six types of problems (as shown in Table 2). For generate problems, the subject had to write a program; for interpret problems, the subject had to describe what the program would do.

The proportion correct response by type of problem is given in the top of Table 3. As can be seen, the control gropu performs well on problems that are very much like the material in the instructional text, e.g., generate-statement and generate-nonloop. However, on problems that require moderate amounts of transfer--e.g., generate-loop and the shorter interpret problems; the model group excels. This difference in the pattern of performance suggests that models enhance transfer performance but not simple retention of presented material. Apparently, the model provided an assimilative context in which novices could relate new technical information in the booklet to a familiar analogy. This assimilative process resulted in a broader learning outcome that supported moderate transfer.

Locus of the effect. One problem with the above study is that the model subjects received more information than controls. Therefore, another series of studies was conducted (Mayer, 1976a) in which all subjects read the same BASIC-like manual, but some subjects were given a concrete model of the computer before reading while others were given the same model after reading the manual.

The proportion correct response by type of problem for the two groups is shown in the bottom of Table 3. As in the previous study, the before group excels on creative transfer to new situations but the after group excels on simple retention of the presented material. Thus, as predicted by assimilation theory, the model serves as an assimilative context for learning only if it is available to the learner at the time of learning.

Recall. The above studies used transfer tests as measures of what is learned under different instructional treatments. In a follow-up study (Mayer & Bromage, 1980), subjects read the manual and were given model either before or after the reading as in the previous study. However, as a test, subjects were asked to recall all they could about certain portions of the manual.

In order to score the protocols, the information in the manual was broken down into idea units. Each idea unit expressed one major idea or action. There were three kinds of idea units in the manual: (1) conceptual idea units related to the internal operation of the computer, (2) technical idea units gave examples of code, and (3) format idea units gave grammer rules. Table 4 gives examples of each type of idea unit.

Table 5 shows the average number of idea units recalled from each category by the two groups. As can be seen, the before group recalls more conceptual information while the after group excels on recall of technical and format information. This pattern is consistent with the idea good retention requires recall of specific code, but good transfer requires understanding of conceptual ideas. Also, the before group included more intrusions about the model and other sections of the manual, suggesting they integrated the information more broadly.

<u>Different language</u>. Although the above results are consistent and were obtained in a long series of studies, their generality is limited by the fact that just one type of language was used. Thus, a follow-up study was conducted (Mayer, 1980a) in which subjects learned a file management language (Gould & Ascher, 1974) either with or without a concrete model. Table 6 lists the eight statements that were described in the instructional manual. Figure 3 shows the concrete model that was used: long-term memory is represented as a file cabinet; the sorting function is represented as an in-basket, out-basket and save basket; temporary memory is represented as an erasable scoreboard; executive control is represented as a list and pointer arrow; output is represented as a message pad. After instruction, all subjects took a transfer test including simple retention-like problems (sort-1) and problems that required putting all of the learned commands together in a novel way (compute-1 and 2). (See Table 7.)

Table 8 gives the proportion correct response by type of problem for the two treatment groups. As in the previous studies, the control group performs best on simple problems like those in the manual, but the model group excels on longer problems that require creative integration. Thus, previous results and conclusion seem to generalize to this new domain.

Ability. The pattern of result described above tended to be strongest for low ability subjects (Mayer, 1975) where ability is defined in terms of Mathematics SAT scores. Apparently, high ability learners already possessed their own useful "models" for thinking about how a computer works, but low ability students would be more likely to lack useful prerequisite knowledge.

Text organization. The pattern of results described above also tended to be strongest when material was poorly organized (Mayer, 1978). Apparently, the model is more useful when material is poorly structured because it helps the reader to hold the information together.

Evaluation. These results provide clear and consistent evidence that a concrete analogical model can have a strong effect on the encoding of new technical information in novices. These results provide empirical support to the claims of DuBoulay & O'Shea (1976, 1978) that allowing novices to "see the works" allows them to encode information in a more coherent and useful way. When appropriate models are used, the learner seems to be able to assimilate each new statement to his or her image of the computer system. Thus, one straightforward implication is: if your goal is to produce learners who will not need to use the language creatively, then no model is needed; if your goal is to produce learners who will be able to come up with creative solutions to novel (for them) problems, then a concrete model early in learning is quite useful. More research is needed in order to determine the specific effects of concrete models on what is learned, and to determine the characteristics of a useful model.

2. ENCOURAGE LEARNERS TO "PUT IT IN THEIR OWN WORDS"

Statement of the Problem

A second technique for increasing the meaningfulness of technical information is elaboration--encouraging the learner to explain the information in his or her own words and to verbally relate the material to other concepts or ideas. Elaboration techniques may influence meaningful learning because they encourage the activation of existing knowledge that is relevant for comprehending the presented material--i.e. elaboration may affect the activation process as indicated by the arrow c in Figure 1.

Example

For example, in our own research, we have taught subjects a simple file management language as described in the previous section (see Tables 6 and 7). In order to encourage subjects to elaborate on the material, we presented questions after each page of the instructional booklet. Table 9 gives examples of "model elaboration questions" which ask the learner to relate the material to a familiar context, and "comparative elaboration questions" which ask the learner to relate one part of the material to another.

Background

There is some evidence that asking subjects to put ideas into their own words during learning can enhance the breadth of learning. For example, Gagne & Smith (1962) found that subjects who were required to give a verbal rationalization for each move as they learned to solve a new problem resulted in longer learning time but better transfer performance than non-verbalizers. Results by Seidel & Hunter (1970) suggest that verbalization per se may not significantly enhance computer programming performance.

More recently, Wittrock (1974) has proposed the "generative hypothesis"--

i.e. learning occurs when the learner actively generates associations between what is presented and what he or she already has in memory. For example, when school children were asked to generate a one-sentence summary for each paragraph in a prose passage, Wittrock (1974) found that recall was nearly double that of a control group. Apparently, when students are encouraged to actively put information in their own words, they are able to better connect new information to existing knowledge.

Elaboration techniques have long been used to enhance learning of paired associates. For example, when students are asked to actively form images or sentences involving word pairs, paired associate recall is greatly enhanced (Bower, 1972; Paivio, 1969). More recently, elaboration techniques have been used in school curricula (see Dansereau, 1978; Weinstein, 1978). Several researchers have argued that students should be given explicit training in "learning strategies"--i.e. how to actively process new material (see O'Neill, 1978).

Transfer. In a typical study, (Mayer, 1980a) subjects read the instructional booklet covering a simple file management language, with some subjects having an elaboration page after each page in the booklet (model elaboration) and others not (control). A second study followed the same procedure but there was a comparative elaboration page after each page for half the subjects.

On a subsequent transfer test, using problems described in Table 7, the control groups performed well on simple retention-like problems but the elaboration groups (both model and comparative) perform better on problems requiring creative transfer. Table 10 shows the proportion correct by type of problem for four treatment groups. Thus, there is evidence that requiring the learners to put technical information in their own words through relating the material to a familiar situation or through making comparisons, results in broader learning.

Recall. In order to assess the generality of these findings, the studies were replicated using recall as the test (Mayer, 1980a). For scoring, the manual was divided into idea units. Some idea units described how the computer operated (conceptual idea units) and others emphasized the grammar and technical aspects of each statement (technical idea units). Table 11 shows the average number of idea units recalled by type for model elaboration, comparison-elaboration, and control groups. As can be seen, the control group tends to recall equal amounts of both types of information, but the elaboration groups tend to emphasize recall of conceptual as compared to technical information. This pattern is consistent with the idea that conceptual emphasis is likely to support transfer performance.

Notetaking. In order to provide further generality, an additional series of studies was conducted (Peper & Mayer, 1978) using a different language (a BASIC-like language) and a different elaboration activity (note-taking). Subjects watched a 20 minute videotape lecture describing seven BASIC-like statements similar to the manual described earlier. Some subjects were asked to take notes by putting the basic information in their own words. Others simply viewed the lecture without taking notes. As a test some subjects were given transfer problems and some were asked to recall portions of the lesson.

As in previous studies, there was a pattern in which note-taking improved performance on far transfer problems but not on simple retention problems. Similarly, there was a pattern in which note-takers performed better on recall of conceptual information but not technical information. These patters were observed for subjects scoring low in Mathematics SAT, but not for high ability subjects. Presumably high ability learners already possess strategies for putting new information into their own words.

Evaluation. Unfortunately, there is no fool-proof way to design elaboration activities. However, it is important to keep in mind that the goal of elaboration is to help the learner be able to describe the key concepts in his or her own words, using existing knowledge. Emphasis on format or grammatical details, and emphasis on errorless verbatim recall of statements will not produce the desired effects. The learner should be able to describe the effects of each statement in his or her own words.

3. ASSESS AND BUILD ON LEARNERS' INTUITIONS

Statement of the Problem

Learners come to the learning situation with certain existing expectations and intuitions about how to interact with computers. For example, since students have experience with conversations in English, they are likely to try to view computer conversations in the same way (Miller & Thomas, 1977; Sackman, 1970). Similarly, since most users are familiar with calculators, they may view interactions with computers in the same way (Mayer & Bayman, 1980; Young, 1980).

Example

For most users, calculators represent the first exposure to interacting with a computational machine. Thus, intuitions that are established may be important for later learning of computer programming languages. For example, consider the keystrokes:

7 + =

If subjects have a conception of incrementing internal registers, they might suppose that this sequence would result in 14 being displayed. However, less sophisticated intuitions might predict that the display would show 7 or 0.

Background

There is a growing interest in using words and logical structures that are similar to everyday English. For example, Ledgard, Whiteside, Singer & Seymour (1980) found that text editing systems that use "natural language" are easier to learn than those that use "computerese" for commands. Similarly, Shneiderman (1980) reports that meaningful or mnemonic variable names may affect programming performance. Finally, there is evidence that branching structures used in BASIC are not as intuitive or as easy to learn as other branching structures (Green, 1977; Mayer, 1976b; Sime, Green & Guest, 1977; Sime, Arblaster & Green, 1977).

More recently, Young (1980) has developed "mental models" of calculators--i.e. representations of the internal components that a learner needs to understand. Scandura, Lowerre & Veneski (1976) have interviewed children who learned to use calculators through "hands on experience". Many develop bizzarre intuitions even though they can use the calculator to solve routine problems. Thus, in order to build on the learners intuitions, and modify them as needed, one must assess what those intuitions are. In other words, the instructor should have techniques for determining the learner's "mental model".

Analysis of Users' Intuitions of Calculator Operations

A series of studies was conducted (Mayer & Bayman, 1980) in order to determine the intuitions that novice and expert users have concerning how pocket calculators operate. The novices were college students with no experience with computers or computer programming, while the experts were intermediate level computer science students. Each subject was given a 4-page questionnaire with 88 problems. Each problem listed a series of key presses and asked the student to predict what number would be in the display, assuming a standard four-function calculator was being used.

The subjects differed greatly with respect to when they thought an expression should be evaluated. For example, consider the problems,

2 + 3

2 + 3 +

2 + 3 + 7

2 + 3 + 7 =

Some subjects behaved as if an expression was evaluated only when an equals was pressed; thus, the answers were 3, 3, 7, 12. Others behaved as if an expression was evaluated as soon as an operator key was pressed, yielding answers of 3, 5, 7, 12. Finally, some subjects behaved as if an expression was evaluated as soon as a number was pressed, giving answers of 5, 5, 12, 12. Results indicated significant differences between experts and novices, with most experts opting for second approach while novices were fairly split among all three approaches. There also were important differences concerning how to evaluate a chain of arithmetic such as, 2 + 3 x 7 =, and how to handle non-standard sequences such as 2 + = + =.

Evaluation. We are just beginning to develop techniques for describing users' intuitions, e.g., users' mental models of computational machines. However, as techniques become available, teachers may use them to diagnose whether students have acquired useful intuitions, and to remediate where needed.

4. PROVIDE TRAINING IN CHUNKING

Statement of the Problem

One technique for making storage of information easier is to form meaningful chunks of schemas (Bransford, 1979). Within the context of computer programming, this means that learners should develop the ability to view a cluster of statements as a single unit that accomplishes some namable goal.

Example

For example, Atwood & Ramsey (1978) suggest that experienced programmers encode a segment such as,

```
    SUM = 0
    DO 1 I = 1, N
         SUM = SUM + (I)
    1 CONTINUE
```

as "CALCULATE THE SUM OF ARRAY X".

Background

There is some evidence that experts and novices in a particular domain differ with respect to how they organize information in memory, with experts using more efficient chunking techniques (Larkin, McDermott, Simon & Simon, 1980). In recent reviews of research on how to teach people to become better problem solvers, Greeno (1980) and Simon (1980) conclude that good problem solving performance requires that the user has large amounts of domain specific knowledge organized into chunks. For example, Simon (1980) estimates that a person needs 50,000 chunks of domain specific knowledge (e.g., such as the example given above) to become an expert.

In a classic study, Chase & Simon (1973) asked subjects to view briefly presented chess board configurations and then try to reconstruct them. Chess masters performed better than less experienced players in reconstructing positions from actual games, but the advantage was lost when random board positions were presented. In an analogous study reported by Shneiderman (1980), experienced and inexperienced programmers were given programs to study. The experts remembered more than the novices when actual programs were presented but not for random lines of code. These findings suggest that experts have a large repertoire of many meaningful chunks, i.e. ways of grouping many lines of code into a single meaningful unit. More recently, Mayer (1979c, 1980b) has suggested that highly used chunks, such as looping structures, should be explicitly taught and labeled as part of instruction. For example, frequent looping structures in BASIC include "repeating a READ", "waiting for a data number", "waiting for a counter", and "branching down".

5. PROVIDE TRAINING IN ANALYSIS OF STATEMENTS

Statement of the Problem

What does it mean to "understand" a statement" In many psycholinguistic theories, comprehension involves relating a statement to its underlying case grammar (see Kintsch, 1974).

Example

In a previous paper (Mayer, 1979c), I have suggested a possible case grammar for BASIC. Each statement may be described as a list of transactions. A transaction consists of an action applied to some object at some location in the computer. For example, the statement, LET X = 5, consists of six transactions:

1. Find the number indicated on the right of the equals.
2. Find the number in the memory space indicated on the left of the equals.
3. Erase the number in that memory space.
4. Write the new number in that space.
5. Go on to the next statement.
6. Do what it says.

Background

An implication of the "transaction" approach is that the same statement names may actually refer to several different types of transactions. For example, we have shown that a counter set LET such as LET X = 5 is different from an arithmetic LET such as LET X = 10/2 (Mayer, 1979c, 1980c). Explicit naming and describing of different types of statements with the same keyword may become a useful part of computer literacy curricula. More recently this approach has been successfully applied to the analysis of commands in "calculator language" (Mayer & Bayman, 1980) and text editor languages (Card, Moran & Newell, 1980).

CONCLUSION

This paper has provided five tentative recommendations, listed in the introduction, for increasing the meaningfulness of computer concepts for novices. Reviews of cognitive research indicate that there is qualified support for the first two recommendations, and that active research is needed concerning the latter three recommendations.

Note

I wish to thank Bob Seidel and Ron Anderson for their useful comments on an earlier version of this manuscript. A more detailed version of this paper is available as a technical report from the author. Much of the work cited in this paper was supported by grant SED77-19875 from the National Science Foundation and grant NIE-G80-0118 from the National Institute of Education.
Requests for reprints should be sent to: Richard E. Mayer, Department of Psychology, University of California, Santa Barbara, CA 93106.

References

Atwood, M. E. & Ramsey, H. R. Cognitive structure in the comprehension and memory of computer programs: An investigation of computer programming debugging. (ARI Technical Report TR-78-A210. Englewood, Colorado: Science Applications, Inc., August, 1978.

Ausubel, D. P. The use of advance organizers in the learning and retention of meaningful verbal material. Journal of Educational Psychology, 1960, 51, 267-272.

Ausubel, D. P. The psychology of meaningful verbal learning. New York: Gruene and Stratton, 1963.

Ausubel, D. P. Educational psychology: A cognitive view. New York: Holt Rinehart & Winston, 1968.

Bower, G. H. Mental imagery and associative learning. In L. Gregg (Ed.), Cognition in Learning and Memory. New York: Wiley, 1972.

Bransford, J. D. Human cognition. Monterey, CA: Wadsworth, 1979.

Bransford, J. D. & Johnson, M. K. Contextual prerequisites for understanding: Some investigations of comprehension and recall. Journal of Verbal Learning and Verbal Behavior, 1972, 11, 717-726.

Brownell, W. A. & Moser, H. E. Meaningful vs. mechanical learning: A study in grade III subtraction. In Duke University research studies in education, No. 8. Durham, N. C.: Duke University Press, 1949, 1-207.

Card, S. K. Moran, T. P., & Newell, A. Computer textediting: An information processing analysis of a routine cognitive skill. Cognitive Psychology, 1980, 12, 32-74.

Chase, W. G. & Simon, H. A. Perception in chess. Cognitive Psychology, 1973, 4, 55-81.

Cook. L. & Mayer, R. E. Effects of shadowing on prose comprehension and problem solving. Memory and Cognition, 1980, 8, in press.

Dansereau, D. The development of a learning strategies curriculum. In H. F. O'Neil (Ed.), Learning strategies. New York: Academic Press, 1978.

Dooling, D. J. & Lachman, R. Effects of comprehension on the retention of prose. Journal of Experimental Psychology, 1971, 88, 216-222.

Dooling, D. J. & Mullet, R. L. Locus of thematic effects on retention of prose. Journal of Experimental Psychology, 1973, 97, 404-406.

du Boulay, B. & O'Shea, T. How to work the LOGO machine. Edinburgh: Department of Artifical Intelligence, Paper No. 4, 1976,

du Boulay, B., O'Shea, T. & Monk, J. The black box inside the glass box: Presenting computering concepts to novices. Edinburgh: Department of Artificial Intelligence, Paper No. 133, 1980.

Gagne, R. M. & Smith, E. C. A study of the effects of verbalization on problem solving. Journal of Experimental Psychology, 1962, 63, 12-18.

Gould, J. D. & Ascher, R. M. Query by non-programmers. Paper presented at American Psychological Association, 1974.

Green, T. R. G. Conditional program statements and their comprehensibility to professional programmers. Journal of Occupational Psychology, 1977, 50, 93-109.

Green, T. R. G. & Arblaster, A. T. As you'd like it: Contributions to easier computing. Shefdield, U. K.: MRC Social & Applied Psychology Unit, Memo No. 373, 1980.

Greeno, J. G. Trends in the theory of knowledge for problem solving. In D. T. Tuma & F. Reif (Eds.), Problem solving and education: Issues in teaching and research. Hillsdale, NJ: Erlbaum, 1980.

Holtzman, T. G. & Glaser, R. Developing computer literacy in children: Some observations and suggestions. Educational Technology, 1977, 17, No. 8, 5-11.

Kintsch, W. The representation of meaning in memory. Hillsdale, NJ: Erlbaum, 1974.

Larkin, J., McDermott, J., Simon, D. & Simon, H. A. Expert and novice performance in solving physics problems. Science, 1980, 208, 1335-1342.

Ledgard, H., Whiteside, J., Singer, A. & Seymour, W. The natural language of interactive systems. Communications of the ACM, 1980, 23, in press.

Mayer, R. E. Different problem-solving competencies established in learning computer programming with and without meaningful models. Journal of Educational Psychology, 1975, 67, 725-734.

Mayer, R. E. Some conditions of meaningful learning for computer programming: Advance organizers and subject control of frame order. Journal of Educational Psychology, 1976, 143-150. (a)

Mayer, R. E. Comprehension as affected by the structure of problem representation. Memory & Cognition, 1976, 4, 249-255. (b)

Mayer, R. E. Different rule systems for counting behavior acquired in meaningful and rote contexts of learning. Journal of Educational Psychology, 1977, 69, 537-546.

Mayer, R. E. Advance organizers that compensate for the organization of text. Journal of Educational Psychology, 1978, 70, 880-886.

Mayer, R. E. Can advance organizers influence meaningul learning? Review of Educational Research, 1979, 49, 371-383. (a)

Mayer, R. E. Twenty years of research on advance organizers: Assimilation theory is still the best predictor of results. Instructional Science, 1979, 8, 133-167. (b)

Mayer, R. E A psychology of learning BASIC. Communications of the ACM, 1979, 22, 589-594. (c)

Mayer, R. E. Elaboration techniques for technical text: An experimental test of the learning strategy hypothesis. Journal of Educational Psychology, 1980, 72, in press. (a)

Mayer, R. E. Ten statement spiral BASIC: From calculator to computer. Encino, CA: Glenco Publishing Co., 1980. (b)

Mayer, R. E. & Bayman, P. An information processing analysis of users' knowledge of electron calculators. Technical Report No. 80-1. University of California, Santa Barbara: Department of Psychology, 1980.

Mayer, R. E. & Bromage, B. Different recall protocols for technical text due to sequencing of advance organizers. Journal of Educational Psychology, 1980, 72, in press.

Mayer, R. E., Larkin, J. H. & Kadane, J. Analysis of the skill of solving equations. Paper presented at the Psychonomic Society, 1980.

Miller, L. A. & Thomas, J. C. Behavioral issues in the use of interactive systems. International Journal of Man-Machine Studies, 1977, 9, 509-536.

O'Neil, H. F. Learning strategies. New York: Academic Press, 1978.

Paivio, A. Mental imagery in associative learning and memory. Psychological Review, 1969, 76, 241-263.

Peper, R. J. & Mayer, R. E. Note taking as a generative activity. Journal of Educational Psychology, 1978, 70, 514-522.

Resnick, L. B. & Ford, S. The psychology of mathematics learning. Hillsdale, NJ: Erlbaum, 1980.

Royer, J. M. & Cable, G. W. Facilitated learning in connected discourse. Journal of Educational Psychology, 1975, 67, 116-123.

Royer, J. M, & Cable, G. W. Illustrations, analogies, and facilitative transfer in prose learning. Journal of Educational Psychology, 1976, 68, 205-209.

Sackman, H. Experimental analysis of man-computer problem-solving. Human Factors, 1970, 12, 187-201.

Scandura, A. M., Lowerre, G. F., Veneski, J. & Scandura, J. M. Using electronic calculators with elementary school children. Educational Technology, 1976, 16, No. 8, 14-18.

Seidel, R. J. & Hunter, H. G. The application of theoretical factors in teaching problem-solving by programmed instruction. International Review of Applied Psychology, 1970, 19, 41-81.

Shneiderman, B. Software psychology: Human factors in computer and information systems. New York: Winthrop, 1980.

Sime, M. E., Arblaster, A. A. & Green, T. R. G. Reducing programming errors in nested conditionals by prescribing a writing procedure. International Journal of Man-Machine Studies, 1979, 9, 119-126.

Sime, M. E. Green, T. R. B., & Guest, D. J. Scope marking in computer conditionals: A psychological evaluation. International Journal of Man-Machine Studies, 1977, 9, 107-118.

Simon, H. A. Problem solving and education. In D. T. Tuma & F. Reif (Eds.), Problem solving and education: Issues in teaching and research. Hillsdale, NJ: Erlbaum, 1980.

Spilich, G. J., Vesonder, G. T., Chiesi, H. L., & Voss, J. F. Text processing of domain-related information for individuals with high and low domain knowledge. Journal of Verbal Learning and Verbal Behavior, 1979, 18, 275-290.

Weaver, F., & Suydam, M. Meaningful instruction in mathematics education. Mathematics Education Reports, 1972.

Weinstein, C. Elaboration skills as a learning strategy. In H. F. O'Neil (Ed.), Learning strategies. New York: Academic Press, 1978.

Wertheimer, M. Productive thinking. New York: Harper & Row, 1978.

White, R. T. & Mayer, R. E. Understanding intellectual skills. Instruction Science, 1980, 9, 101-127.

Wittrock, M. C. Learning as a generative process. Educational Psychologist, 1974, 11, 87-95.

Young, R. M. Surrogates and mappings: Two kinds of conceptual models for pocket calculators. Paper presented at conference on Mental Models, La Jolla, California, 1980.

Table 1

Seven Statements Used in BASIC-like Instructional Booklet

Name	Example
READ	P1 READ (A1)
WRITE	P2 WRITE (A1)
EQUALS	P3 A1 = 88
CALCULATE	P4 A1 = A1 + 12
GOTO	P6 GO TO P1
IF	P5 IF (A1 = 100) GO TO P9
STOP	P9 STOP

Table 2

Examples of Six Types of Test Problems for a BASIC-like Language

<u>Generation-Statement</u>

Given a number in memory space A5, write a statement to change that number to zero.

<u>Interpretation-Statement</u>

```
A5 = 0
```

<u>Generation-Nonloop</u>

Given a card with a number on it is input, write a program to print out its square.

<u>Interpretation-Nonloop</u>

```
P1  READ (A1)
P2  A1 = A1 * A1
P3  WRITE (A1)
P4  STOP
```

<u>Generation-Looping</u>

Given a pile of data cards is input, write a program to print out each number and stop when it gets to card with 88 on it.

<u>Interpretation-Looping</u>

```
P1  READ (A1)
P2  IF(A1 = 88)  GO TO P5
P3  WRITE (A1)
P4  GO TO P1
P5  STOP
```

Table 3

Proportion Correct on Transfer Test by Type of Problem for Model vs. Control Groups, and Before vs After Groups

	Generation			Interpretation		
	Statement	Nonloop	Looping	Statement	Nonloop	Looping
Model vs. Control						
Model	.63	.37	.30	.62	.62	.09
Control	.67	.52	.12	.42	.32	.12
Before vs. After						
Before	.57	.50	.20	.47	.63	.17
After	.77	.63	.13	.27	.40	.17

Note. For model vs. control, n = 20 per group; interaction between treatment and problem type, $p < .05$.

For before vs. after, n = 20 per group; interaction between treatment and [illegible]blem type, $p < .05$.

Table 4

Examples of Test Problems for a File Management Language

Sort 1 List the owners' names for all cars weighing 3000 pounds or more.	FROM AUTOMOBILE FOR WEIGHT IS CALLED 3000 OR MORE LIST NAME
Sort 2 List the owners' names for all late model green Fords.	FROM AUTOMOBILE FOR YEAR IS CALLED 1976 OR MORE AND FOR COLOR IS CALLED GREEN AND FOR MAKE IS CALLED FORD LIST NAME
Count How many cars are registered in Santa Barbara County?	FROM AUTOMOBILE FOR HOME COUNTY IS CALLED SANTA BARBARA COUNT LIST COUNT
Compute 1 What is the average current value of all cards?	FROM AUTOMOBILE COUNT TOTAL CURRENT VALUE LET TOTAL ÷ COUNT BE CALLED AVERAGE LIST AVERAGE
Compute 2 What percentage of 1977 cars are Chevrolets?	FROM AUTOMOBILE FOR YEAR IS CALLED 1977 COUNT LET THIS BE CALLED COUNT 1 AND FOR MAKE IS CALLED CHEVROLET COUNT LET THIS BE CALLED COUNT 2 LET COUNT 2 ÷ COUNT 1 BE CALLED AVERAGE LIST AVERAGE

Table 5

Average Number of Recalled Idea Units for the Before and After Groups

	Idea Units			Intrusions		
	Technical	Format	Conceptual	Inappropriate	Appropriate	Model
Before	5.0	1.9	6.6	1.5	1.3	3.1
After	6.0	2.9	4.9	2.5	.8	.5

Note. N = 30 per group; interaction between treatment and type of score, $p < .05$.

Table 6

Eight Statements Used in File Management Language Booklet

Name	Example
FROM	FROM AUTOMOBILE
FOR	FOR WEIGHT IS CALLED 3000 OR MORE
AND FOR	AND FOR COLOR IS CALLED GREEN
OR FOR	OR FOR MAKE IS CALLED FORD
LIST	LIST NAME
COUNT	COUNT
TOTAL	TOTAL CURRENT VALUE
LET	LET TOTAL ÷ COUNT BE CALLED AVERAGE

Table 7

Example of Conceptual, Format, and Technical Idea Units

Type	Idea Unit
Technical	READ is one kind of statement.
Format	The format is READ ( ).
Format	An address name goes in the parenthesis.
Conceptual	An address name is a space in the computer's memory.
Conceptual	There are 8 memory spaces.
Technical	The spaces are called A1, A2 ....
Technical	An example is, READ (A2).
Conceptual	First, the computer checks the number from the top data card.
Conceptual	Then, that number is stored in space A2.
Conceptual	The previous number in A2 is destroyed.
Conceptual	Then the data card is sent out of the computer.
Conceptual	This reduces the pile of data card by 1.
Conceptual	Then, go on to the next statements.

Table 8

Proportion Correct on Transfer Test for Model and Control Groups--
File Management Language

	Type of Test Problem				
	Sort-1	Sort-2	Count	Computer-1	Compute-2
Model	.66	.66	.63	.58	.45
Control	.63	.44	.43	.33	.22

Note. N = 20 per group; treatment x problem type interaction, p < .07.

Table 9

Example of the Elaboration Exercise in the Programming Text

Model Elaboration

Consider the following situation. An office clerk has an in-basket, a save basket, a discard basket, and a sorting area on the desk. The in-basket is full of records. Each one can be examined individually in the sorting area of the desk and then placed in either the save or discard basket. Describe the FOR statement in terms of what operations the clerk would perform using the in-basket, discard basket, save basket, and sorting area.

Comparative Elaboration

How is the FOR command like the FROM command?

How is the FOR command different than the FROM command?

Table 10

Proportion Correct on Transfer Test by Type of Problem for Model Elaboration vs. Control Groups and Comparative Elaboration vs. Control Groups

	Type of Test Problem				
Model vs. Control	Sort-1	Sort-2	Count	Compute-1	Computer-2
Model Elaboration	.65	.58	.64	.64	.45
Control	.66	.64	.41	.38	.27
Comparative Elaboration					
Comparative Elaboration	.90	.90	1.00	.75	.55
Control	.90	.90	.65	.65	.25

Note. For model elaboration vs. control, n = 20 per group; treatment x problem type interaction, p < .05. For comparative elaboration vs. control, n = 13 per group; treatment x problem type interaction, p < .05. Data is for interpretation problems only, for comparative and control groups.

Table 11

Average Number of Recalled Idea Units for Model Elaboration, Comparative Elaboration and Control Groups

	Type of Idea Units	
	Technical	Conceptual
Model Elaboration	5.3	13.9
Comparative Elaboration	9.4	14.1
Control	7.5	7.5

Note. N = 20 per group; treatment x type interaction, $p < .05$ for low ability subjects.

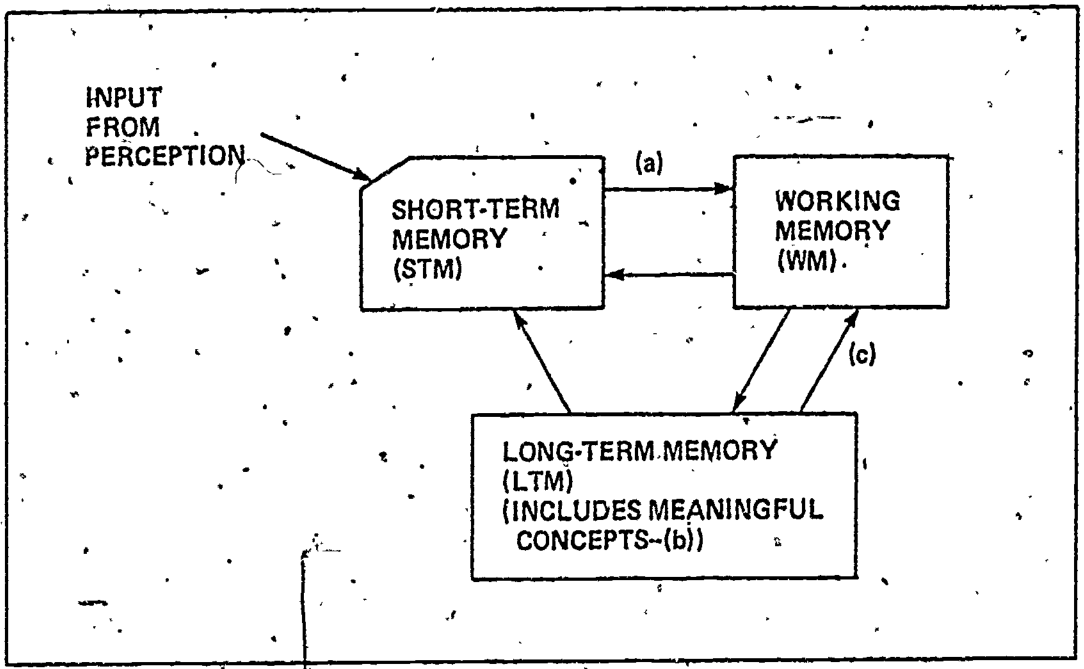

Figure 1: Some information processing components of meaningful learning. Condition (a) is transfer of new knowledge to WM. Condition (b) is availability of assimilative context in LTM. Condition (c) is activation and transfer of old knowledge to WM.

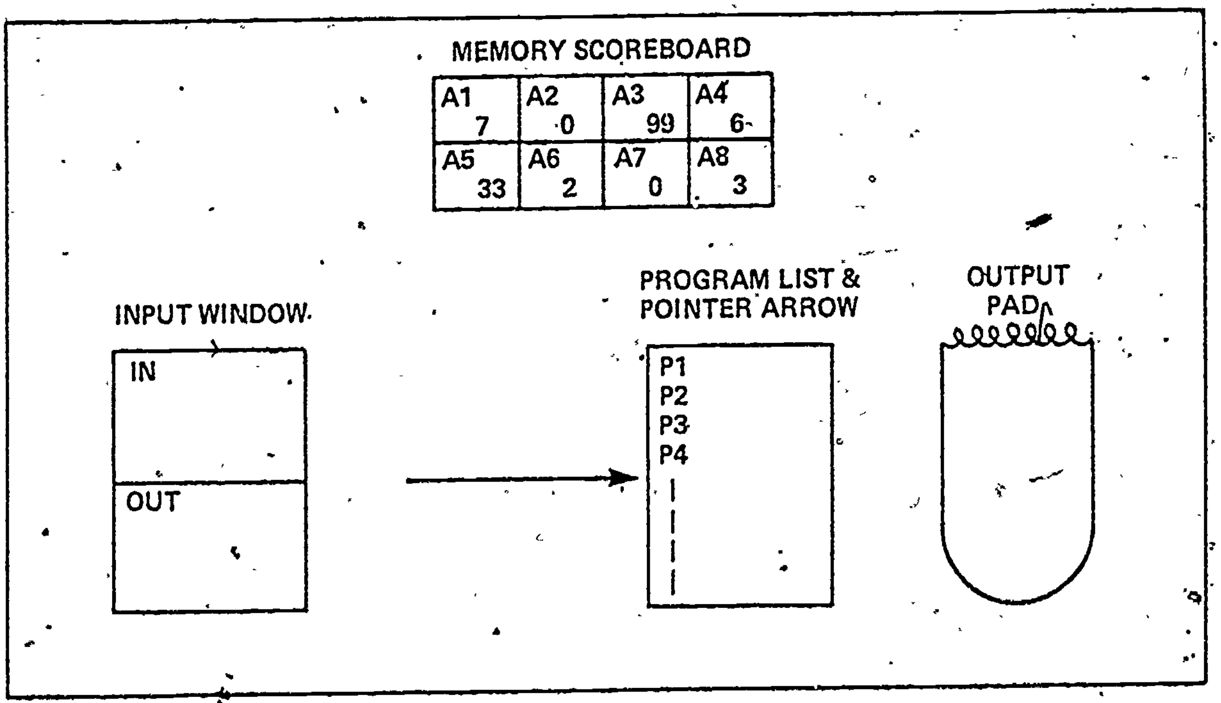

Figure 2: A concrete model of the computer for a BASIC-like language.

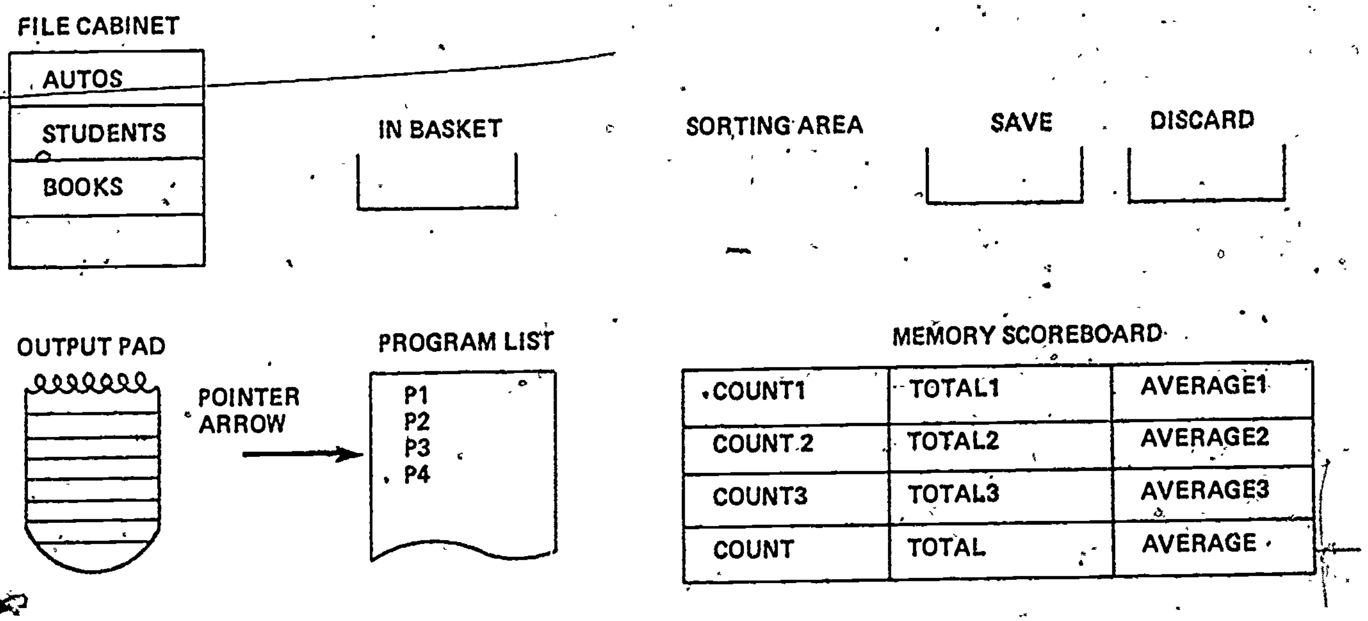

Figure 3. A concrete model of the computer for a file management language.

CHAPTER 2

DIAGNOSIS AND REMEDIATION OF BUGS IN CALCULATOR LANGUAGE

Note

The material in this chapter was published as the following article:

Mayer, R. E., & Bayman, P. A psychology of calculator languages: Describing computer concepts to novices. Communications of the Association for Computing Machinery, 1981, 24, 511-520.

Abstract

This paper presents a framework for describing users' knowledge of how a simple four-function calculator operates. Data are summarized concerning differences among novices and experts in their conceptions of "what goes on inside the calculator" for various sequences of button presses. Individual differences include: different view on when an expression is evaluated, different procedures for evaluating a chain of arithmetic, and different rules for evaluating unusual sequences of key presses.

Key Words and Phrases: calculator, learning, instruction, psychology

CR Categories: 1.50, 2.12, 4.20

A major goal of this article is to provide a framework for describing users' knowledge of calculator language, i.e., users' intuitions concerning the underlying logic of a simple four-function calculator when a series of buttons are pressed. Another major goal is to use this technique for pinpointing some of the differences among actual users in their knowledge of calculator language. The first section of this paper provides a rationale and a brief literature review. The second section describes the transaction approach for analyzing calculator language. The third section summarizes a study of individual differences among users in their knowledge of calculator language. The final section provides a summary and a set of tentative recommendations.

Rationale

Performance versus competence. A basic idea in this article is that the traditional distinction between performance and competence can and should be applied to users' learning of calculator language. Performance, of course, refers to what the user can do, such as compute answers for a class of problems; competence refers to what the user knows, such as the user's mental model of the calculator.

It is possible for two users to both give identical answers to simple arithmetic computation problems, but possess vastly different underlying knowledge of calculator language. For example, in pilot studies we found that a subject believed the calculator had the answer for each problem already stored in memory. This subject would claim that to find the answer for 22 x 114, the calculator simply "looks up" the answer for that problem in its memory. Another subject assumed that the calculator used "internal registers," and followed certain "control procedures." For 22 x 114, the calculator would store the numbers 22 and 114 in memory, and would use the multiplication algorithm to work on them. These two subjects seem to have had different "mental models" for the calculator. Thus, for a complete description of "what is learned" by different users, we must be able to describe the users' competence as well as their performance. Similarly, Greeno [3] has argued for emphasis on "cognitive objectives" of instruction rather than focusing solely on "behavioral objectives" of instruction.

Black box versus glass box. A second important distinction concerns learning by memorizing versus learning by understanding [15]. When we apply this distinction to the learning of calculator language, we can point to the difference between the "black box approach" and the "glass box approach." In the black box approach to learning calculator language, the user focuses only on the external features of calculator language--you put in a sequence of key presses and out comes the answer as if by magic. The operations inside the calculator are hidden from the user, forcing the user to treat the calculator as a black box that cannot be understood. A user who learns by the black box method is forced to memorize sequences of key presses for each type of problem, without understanding what the key presses actually mean. For example, some manuals describe how to use a constant. Let's say you want to multiply a set of numbers by 2.3. The manual may instruct you to enter the sequence 2.3 x x; then, for any number you want multiplied by 2.3 you just enter that number followed by an equals (=). Although memorized procedures, such as the "constant" sequence, may work in the sense that they generate the desired answer, the user is not able to relate the sequence of key strokes to an understanding of what goes on inside the calculator. DuBoulay & O'Shea [1] have noted a similar phenomenon with respect to children learning LOGO; some users act as if the internal operations of the machine are hidden and not understandable.

In the glass box approach [2], the user is able to see how a sequence of key strokes is related to change in the internal state of the calculator, and how these changes are related to the final answer. Each command--in this case, each key stroke--results in some change inside the calculator and these changes can be described and understood. For example, the user who learns by the glass box approach may be able to describe why the "constant" procedure works, by describing the nature of internal displays and incrementing operations.

The level of description of events in the glass box approach need not, indeed should not, be at the "blood and guts" level. By this we mean that users need not become electronic experts. There is an appropriate level of description that Young [16] refers to as the user's "mental model" of the calculator: "For an interactive system to be satisfactory, it is important that its intended users be able to form a model of the system which enables them to predict its behavior."

DuBoulay, O'Shea & Monk [2] have suggested that novices should be exposed to a "notational machine" -- i.e. "an idealized model of the computer implied by the constraints of the programming languages" and which is analogous to "other mechanisms with which the novice is more familiar". As an example, duBoulay & O'Shea [1] have developed a "LOGO machine" to represent the internal actions that occur for LOGO statements. Further, duBoulay, O'Shea & Monk [2] have offered two important properties for selecting a model that clarifies the hidden operations of a language: (1) simplicity--there should be a "small number of parts that interact in ways that can be easily understood", and (2) visibility--novices should be able to see "selected parts and processes of this notational machine in action".

As an example, let's suppose that we want students to learn how to solve simple arithmetic problems with a calculator. We could give them plenty of hands-on experience, without any guidance about what goes on inside the calculator, until they were all able to solve simple problems. However, Scandura, Lowere, Veneski & Scandura [13] found that students who taught themselves to use calculators often developed bizzare intuitions; one student, for example, concluded that the plus (+) and equals (=) keys did nothing since they caused no visible change in the display. Instruction that emphasises the understanding of how the machine operates on a sequence of button presses might provide a better base on which to build <u>further computer</u> concepts.

What are the benefits of instruction that foster glass box learning rather tnan black box learning? Past research by Gestalt psychologists [15] suggest that learning by understanding, such as in the glass box procedure, should lead to superior long-term retention and superior transfer to novel problems. In addition, the glass box approach may influence attitudes concerning the "understandability" of computers and calculators. Although there is promising support for these assertions in studies of how novices learn simple programming languages [4, 5, 6, 7, 9], much more research is needed concerning the role of glass box instruction for calculators.

<u>Computer literacy</u>. The previous sections have presented two ways of describing "what is learned" (i.e., performance vs. competence) and two ways of teaching how to use calculators (i.e., black box vs. glass box). Why is it important to focus on how students learn and represent knowledge about calculators? The reason is that calculators (as well as electronic games) usually involve a user's first exposure to a computational machine and a language. Thus, calculators could provide the first step in the development of a user's computer literacy--the understanding of how to interact with computational machines.

In addition, calculators have become a part of society, infiltrating the home, work, and school lives of ordinary people (10). In our schools, for example, teachers [11] have recognized calculators as necessary tools in our society: "The National Council of Teachers of Mathematics recommends that mathematics programs take full advantage of the power of calculators and computers at all grade levels." However, in spite of the potential for using calculators as the first step towards computer literacy, there is also the potential that they will be used as tools whose operations must be blindly memorized. For example, duBoulay, O'Shea & Monk (2) recently pointed out: "The manuals accompanying certain makes of pocket calculators make no attempt to explain the reason why given sequences of button presses carry out the given computations. The user must follow the manual's instructions blindly because it is difficult for him to imagine what kind of underlying machine could be inside that demands these particular sequences of presses. During the course of a calculation, he has to guess the current state of the device.... because the device gives little or no external indications of its internal state."

Unfortunately, the research community has been very slow in providing information that would be useful in this impending calculator-curriculum revolution. For example, most experimental studies have been concerned with whether using calculators in the classroom affects overall achievement and/or attitude in mathematics [see 12, 14]; but as Roberts [12] recently concluded, "the research literature offers no guidance" concerning how to incorporate calculators into school curricula.

The development of a theory of how users conceptualize calculator language has implications for the design of calculator languages, for instructional procedures, and for integration of calculators into school curricula. This paper is based on the idea that calculators are here to stay, that large numbers of ordinary (non-programmers) people will be using them, and that calculators provide most users with their first introduction to computer concepts. Such users will inevitably develop attitudes and approaches to human/computer interaction in the course of learning to use their calculator even if the users are self-taught. This paper provides some information that may be relevant to understanding what intuitions individual users have about calculators. Thus, this paper is a start towards the goal of helping users to make the most of their calculators and computers in general.

A Transaction Analysis of Calculator Language

The goal of this section is to develop an appropriate level of describing what happens inside the calculator for each type of key press, based on DuBoulay, O'Shea & Monk's, [2], criteria of simplicity and visibility. In particular, this section, applies the transaction approach to the operating system[1] of electronic calculators, or to what can be called calculator language. The goal is not to provide a formal, mathematical representation of the calculator's operating system, but rather to provide an idealized model of the calculator that can be used to describe users' knowledge and to help novices understand calculator language. It should be pointed out that the transaction approach may serve both as (1) a descriptive model of the users' knowledge of calculator language, and (2) a prescriptive model for curriculum development. This paper presents data concerning the first implementation, but also suggests implications concerning the second.

For purposes of this analysis we assume that each user's conception of calculator language can be specified as a set of productions, or condition-action pairs. The condition refers to some key press (i.e., some command) and the action refers to one or more transactions (i.e., an operation applied to an object at a location in the calculator). Thus, the transaction approach involves locating the transaction (or list of transactions) that a user associates with a given command.

A previous paper [6] summarized a conceptual analysis of BASIC that emphasized "transactions" for describing the language in a simple and visible way. A model was constructed that consisted of a ticket window to represent the input function, a note pad to represent the output function, a memory scoreboard to represent the memory function, a scratch pad to represent the logic and arithmetic function, a shopping list with pointer arrow to represent executive control. Each elementary BASIC statement was described as a list of transactions, and each transaction consisted of some operation applied to some object at some location in the computer. Using a small collection of transactions it was possible to describe each of the elementary BASIC statements. Further, there is substantial evidence that instruction in BASIC which emphasizes the transaction level of description, results in superior performance in creative program writing and interpreting written programs [4, 5, 7, 9].

The relevant conditions (or commands) for the present analysis are based on pressing keys on the calculator's keyboard. The keys relevant to a very simple four-function calculator are number keys (i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9), operation keys, (i.e., +, −, x, ÷), equals key (=), decimal key (.) and clear key (CLR). For the present study we assumed that the calculator used arithmetic logic (rather than algebraic or reverse Polish notation) and we focused on only three number keys (i.e., 2,3, and 7), two operation keys (i.e., + and x), and the equals key (=).

Thus, at first blush it seems the basic conditions are each of the single key presses, such as pressing a number key, pressing an operation key, and so on. However, interviews with users suggest that key presses have different "meanings" depending on the immediately preceding key press; for example, pressing a plus key after pressing a number key has a different effect than pressing a plus key after pressing an equals key, for some users. Thus, the conditions (or user commands) can be listed as some key being pressed given that some key was pressed immediately before. For example, typical commands in the present analysis are listed in Table 1. There are certainly many other possible commands, but we have focused on this set of 16 elementary calculator commands as an example.

Insert Table 1 about here

To describe the actions that occur for any command, the transaction approach [6] requires that we specify the triplet of location, object, and operation. The possible locations within the calculator are:

(1) Display -- The external display normally consists of at least eight spaces, where a place can hold one digit. The display fills from the right.

(2) Register -- An internal register is inside the calculator and consists of a series of subregisters that hold individual numbers and operators. Expressions are held in the order of input, with the first number of the left, followed by first operator, and with new numbers and operators entered to the right of existed filled subregisters.

(3) Keyboard -- The external set of keys includes number, operation, and equals keys.

The possible objects include:

(1) Numbers -- A number is any single or multiple digit sequence such as 2, 14, or 156.

(2) Operation -- An operation is a mathematical symbol for some arithmetic computation such as addition (+) or multiplication (x).

(3) Expression -- An expression is a sequence consisting of numbers and operators such as, 2 + 3 or 2 + or 2 + 3 x 7.

Some operations that are relevant to computer language are:

(1) Find -- Locate a particular object; e.g., find a number that was just entered from the keyboard.

(2) Destroy -- A number or expression is erased from the display or register; e.g., when you press the equals key the previous number in the display is erased (and replaced with a new one).

(3) Create -- A number or expression is placed in a display or register; e.g., when you press a number key that number appears in the display.

(4) Evaluation -- An expression from the register is converted into a single number using the rules of arithmetic; e.g., the evaluation of 3 + 2 is 5. (For the current discussion, evaluation of a number or numbers followed by an operation is the number; i.e., evaluation of 3 is 3 or evaluation of 3 + is 3). It is also possible to evaluate expressions from the register and display

together; for example, the evaluation of 2 + in the register and 3 in the display may be 5.

Table 2 gives a summary of some typical actions that might occur with the calculator. Each is expressed as a list of transactions; for example, D = R consists of four separate transactions, while D = D requires only one. A user's conception of what a particular command means can be expressed as a production; for example, the production,

P2 If # after + Then D = # and R = "R + #"

means that when number key is pressed after a plus key, the user assumes that the calculator executes the four transactions for D = # and the three transactions for R = "R + #" as listed in Table 2. Thus, for the sequence 7 + 3, when the 3

Insert Table 2 about here

---

key is pressed the display is changed to 3 and the register's expression is changed to "7 + 3". A user's intuitions concerning calculation language can thus be expressed as a list of productions such as the one given above.

Empirical Studies

There has not been adequate research concerning how students come to understand the operation of calculators. As noted earlier, almost all behavioral research concerning calculators has been directed at the gross issue of whether the availability of calculators in the classroom has any effect on mathematics achievement or attitude [see 11, 14]. The present study addresses a different issue, namely what types of hypotheses do people have concerning how calculators operate. The goal of this research is to determine whether the transaction approach can be successfully used as a framework for describing differences in users' knowledge. The goal of these studies is not to test the transaction approach as a "theory," since it is used here only as a framework for describing what is learned. For more detail concerning the methodology and data analyses see Mayer & Bayman [8].

Method. Our study involved 33 college students who had no computer programming experience ("novices") and 33 college students who were enrolled in advanced programming courses ("experts")[2]. Subjects participated in order to fullfill a course requirement. Each student was given a four page questionnaire with 88 problems. Each problem listed a series of key presses and asked the student to predict what number would be in the display, assuming a standard four-function calculator was being used. In addition, subjections were given a questionnaire asking how many hours a week they used a calculator, how many years they had been using a calculator, what kind of calculators they know, and which calculator model(s) they owned, if any.

Standard sequences: When to evaluate. Our subjects differed greatly with respect to when they thought an expression should be evaluated. For example, consider the sequence,

2 + 3

Some subjects answered "3" and some answered "5". Those who gave 5 seem to be using what we called "immediate evaluation" for # after +. Whenever a number key is pressed after a plus key, the entire expression is evaluated and displayed. However, those who gave 3 as an answer seem to be using "delayed evaluation" for # after +. They wait for some other key press (such as an equals or a plus or a multiply) before they evaluate and display. How would a subject predict the calculator would respond to

2 + 3 + 7

The answer was 12 for the "immediate evaluators" and 7 for the subjects who relied on delayed evaluation for # after +. Our subjects were very consistent, although experts were significantly more consistent than novices in judgments such as these.

Now consider the problem,

2 + 3 +

Some subjects gave 5 as the answer while others gave 3. Those who gave 3 act as if there is "delayed evaluation" for + after 3. For those who gave 5 as an answer, if they also gave 5 as an answer to problems like 2 + 3, they are not evaluating for + after # either; however, if they gave 3 as an answer for 2 + 3, then they seem to opt for "immediate evaluation" for + after #. Similarly, a sequence like,

2 x 3 +

results in 6 for subjects who rely on immediate evaluation for + after #, but in 3 for those who rely on "delayed evaluation" for + after #. (Note that if our subject relies on immediate evaluation for # after x then the answer will also be 6.)

Finally, consider the problem,

2 + 3 =

All subjects gave 5 as an answer. Or consider the problem,

2 + 3 + 7 =

All subjects gave 12 as an answer. However, if our subjects were delayed evaluation for # after + and delayed evaluation for + after # then we know they wait for an equals sign before they evaluate, thus, these subjects would opt for "immediate evaluation" for = after #.

Based on a systematic analysis of our subjects' performance, as suggested in the examples above, we noted three basic strategies for determining when to evaluate an expression--for a number key, or for an operation key, or for an equals key. These conceptions are summarized in Table 3. As can be seen, the consensus of the experts is that a calculator should evaluate when an operation key is pressed after a number, as is common in most but not all calculators. Novices tend to have much more diverse conceptions, and are significantly different from experts.[3] It may also be pointed out that we found no relation between the conceptions of our subjects and the operating systems of their own calculators, nor between the conceptions of our subjects and the amount of time spent each week with a calculator.[4]

Insert Table 3 about here

Standard sequences: Chains of arithmetic. How would you predict a calculator would answer,

2 + 3 x 7 =

How about the problem,

2 x 3 + 7 =

Our subjects varied with respect to how they evaluated a chain of arithmetic. The vast majority of subjects executed the operations in order from left to right, yielding answers of 35 and 13 respectively for the above problems. Some subjects tended to opt for multiplication being carried out before addition, yielding answers of 23 and 13 respectively. Some subjects tended to opt for addition being carried out before multiplication, yielding 35 and 20 respectively. Some subjects opted for carrying out the second operation first, yielding 23 and 20 respectively. Finally, some subjects simply ignored all but the last computation, yielding 21 and 10 respectively. Table 4 summarizes the major strategies for evaluating a chain, based on an analysis of each subject's performance on several problems. As can be seen, most subjects opted for left-to-right evaluation of a chain, although a substantial minority of experts assumed multiplications were carried out before addition. This procedure is characteristic of some sophisticated calculators and computer commands.

Insert Table 4 about here

Non-standard sequences: Equals after operator. The foregoing two sections demonstrated that there are considerable differences among subjects' interpretations of calculator operations even for standard sequences of key strokes. A standard sequence is defined as one that begins with a number and in which an operator (like + or x) or equals (=) may only follow a number. In the present section we explore subjects' conceptions of how the calculator responds to non-standard sequences of key strokes. A non-standard sequence violates the above "grammatical rule of arithmetic" by having two or more operators (+ or x) and/or equal sign (=) in sequence. Users' predictions concerning non-standard sequences are useful because they allow us to diagnose users' conceptions of the internal operation of the calculator.

For example, consider the sequence,

7 + =

or, consider the sequence,

7 x =

How do subjects interpret the calculator's operations? Some subjects assume that a non-standard sequence results in the display being reset; for example, if resetting the display means setting it to zero then subjects give 0 as the answer to the above problems. Another version of the reset strategy is to assume that the calculator will show an E in the display, or that it will flash on and off. A second group of subjects act as if the calculator simply ignores the non-standard sequence; in this case, the calculator display has 7 in it for each of the above sequences. Finally, a third major group acts as if a number has been inserted between the operator and the display; e.g., they treat 7 + = as 7 + 7 = and give an answer of 14, or they treat 7 x = as 7 x 7 = and give an answer of 49. We call these subjects "incrementing display" subjects because they act as if the number in the display is added to the number in the internal register. There are several variations on the incrementing display strategy; for example, 2 + 3 + = can result in 10 or in 8 depending on the subject's conception of when evaluation occurs.

Table 5 summarizes these three major conceptions of what happens when equals follows an operation; as can be seen, the strategy of ignoring the non-standard sequence is the most common but experts are far more likely to opt for the incrementing display conceptualization. The incrementing procedure is a feature of some more sophisticated calculators and reflects a more sophisticated understanding of internal registers.

Insert Table 5 about here

<u>Non-standard sequences: Two consecutive operators</u>. Another non-standard sequence is to have two consecutive operators, such as,

2 + + =

2 x x =

The same strategies were obtained as in the previous section. One subject thought the display would be reset, e.g., the answers would be 0 for each problem. Some subjects ignored the non-standard sequence; thus the display would say 2 for each problem. For example, 2 + + was treated as if it was 2 + and, hence, 2 would be displayed. Finally, some subjects used an incrementing strategy; e.g., 2 ++ = could be interpreted as 2 + 2 + 2 = thus yielding an answer of 6, and 2 x x = could be interpreted as 2 x 2 x 2 = yielding an answer of 8. A variation of this strategy is to treat 2 + + = as 2 + 2 = 4 and 4 + 4 = 8 yielding an answer of 8; similarly 2 x x = is treated as 2 x 2 = 4 and 4 x 4 = 16 yielding an answer of 16. These differences may be formalized in terms of how the internal registers are evaluated and used (see Mayer & Bayman, 8). Table 6 summarizes these strategies, and shows that most subjects opted for ignoring the non-standard sequence, but experts were far more likely to conceive of incrementing operations. Since incrementing is a feature of more sophisticated operating systems, this difference between experts and novices is sensible.

Similar results were obtained for sequences such as,

2 x + 3 =

2 + x 3 =

Most subjects ignored the first operator, yielding answers of 5 and 6 respectively. Some subjects reset the display, often yielding an answer of 0 for each problem. Some subjects used the incrementing strategy, e.g., with answers of 7 and 12, respectively. There was also a subject who ignored the second operator, yielding answers of 6 and 5, respectively; and there was a subject who preferred multiplication to addition, yielding answers of 6 to both problems. The proportions of ignore, reset, and increment conceptions for novices and experts were quite similar to those shown in Table 6.

Insert Table 6 about here

<u>Non-standard sequences: Operator after equals</u>. Suppose the following key strokes were entered,

2 x = x

Subjects who use the ignore conception of non-standard sequences act as if this sequence is 2 x, thus the answer is 2. Subjects who use the reset strategy give 0 as an answer. Subjects who use an increment strategy, give answers like 8, 16 or 4 depending on the particular kind of incrementing system and the subject's conception of when an expression is evaluated. Table 7 summarizes these three strategies and shows that while most subjects rely on the ignore conception, a substantial minority of experts rely on increment strategies and a substantial minority of novices rely on a reset strategy.

Insert Table 7 about here

Production systems. One goal of the study was to formally describe the intuitions of each subject as a list of 13 productions, i.e., 13 condition-action pairs.[4] The left side of Table 8 (or 9) lists the 13 conditions that were present in the 88 problems we asked subjects to solve. For example, # after + means "pressing a number key after pressing an equals key" such as the last two keystrokes in the sequence 2 + 3. The preceding sections have summarized the different possible actions in general terms. The right side of Table 8 (or 9) gives the actions that may be associated with each condition. Actions are indicated as changes in the display (represented as D) or in the register (represented as R).

Table 8 represents one of our novices.[6] The subject evaluates only when an equals key is pressed (as indicated by P7), but does not evaluate an expression when a number key is pressed (as in P2 and P10) nor when an operation key is pressed (as in P4 and P11). The subject evaluates an arithmetic chain in order from left-to-right; thus, for each action involving eval the procedure is left-to-right. The subject ignores all non-standard sequences (as indicated in P5, P6, P8, P12, P13, P14, P15, P16). For example, on the problem,

2 + 3 + 7 =

the subject begins by setting D = 2, R = 2. Then for the + key, P4 says that D = 2, and R = 2 +. For the 3 key, P2 says that D = 3 and R = 2 + 3. For +, P4 says D = 3, R = 2 + 3 +. For 7, P2 says that D = 7 and R = 2 + 3 + 7. Finally, when = is pressed, P7 says D = 12 and R = 12. As another example, consider the problem

7 + + =

First, D = 7 and R = 7. Then when the first + is pressed, P4 results in D = 7 and R = 7 +. Then, when the second + is pressed P5 results in no change so D = 7 and R = 7 +. Finally, when = is pressed, P8 results in evaluation of 7 + which is 7; thus D = 7 and R = 7.

Table 9 represents one of our experts. The subject evaluates when an operation or an equals key is pressed (as in P4 and P11) rather than waiting for an equal key to be pressed (as in P7), but does not evaluate for pressing a number key (as in P2 and P10). The subject evaluates an arithmetic chain by performing multiplication before addition; thus for each action involving eval the procedure is multiply before add. The subject uses an incrementing procedure for most non-standard sequences (as indicated in P5, P6, P8, P12, P13, P14, P15, P16). For example, on the problem

2 + 3 + 7 =

the subject begins by setting D = 2 and R = 2. Then for the + key, P4 says that D = 2 and R = 2 +. After 3 is pressed, P2 says D = 3, R = 2 + 3. After + is pressed, P4 says that D = 5 and R = 5 +. For 7, P2 yields D = 7, R = 5 + 7. Finally, when = is pressed P7 yields D = 12, R = 12. As another example consider the problem,

7 + + =

First, we begin with D = 7 and R = 7. For the first +, P4 results in D = 7 and R = 7 +. For the second +, the number in the display (7) is added to the value in the register (7) to yield a value of 14 so D = 14 and R = 7. For the equals, the number in the display (14) is added to the number in the register (7) to yield 14 so D = 21 and R = 7.

Insert Tables 8 & 9 about here

General Summary and Recommendations

The results show that even though people are able to use their calculators to solve arithmetic problems, there are important individual differences in people's understanding of calculator language. In this paper, we have summarized differences in people's conception of when to evaluate an expression (immediately when a number key is pressed, when an operation key is pressed, or when an equals key is pressed), how to evaluate an arithmetic chain (left-to-right, multiplication-before-addition, etc.), how to evaluate non-standard sequences (ignore, reset, and increment).

In addition, there was a tendency for experts to differ from novices in the following ways. (1) Experts were more consistent than novices. (2) Experts tended to evaluate expressions when an operator key was pressed more than novices. (3) Experts tended to evaluate ultiplication-before-addition in a chain more than novices. (4) Experts tended to increment the display for non-standard sequences more than novices. Thus, the present paper provides some evidence that it is possible to describe user's conception of how calculator language works; in fact, in another paper [8] we provide production model representations for each subject. The fact that people have different conceptions of calculator operation, and that experts tend to develop more sophisticated ideas than novices, have implications for the design of calculator operating systems and instruction.

In a sense, this paper has been a plea for the use of "cognitive objectives" as well as "behavioral objectives" in users' learning of calculator languages. We need to be able to specify what we want the user to <u>know</u> about how the language works, as well as what we want the user to be able to <u>do</u>. The transaction approach provides a technique for describing the knowledge that a user currently possesses, and the knowledge that we would like the user to acquire. One implication of this approach that warrants further study is that explicit training in the objects, locations, and operations (perhaps using a concrete model of the calculator) will enhance development of our cognitive objectives. The data presented in this paper are preliminary, but they provide clear evidence that people's knowledge can be described (based on simple prediction tests) and there are large individual differences among users in what they "know" about calculator language.

The following recommendations are based on the idea that there should be as close a match as possible between the user's conception of how the calculator should operate and the actual operating system of the calculator. Each recommendation should be viewed as a tentative hypothesis that is subject to much future research, rather than a fact that has been established through existing research.

(1) <u>Choose a calculator that corresponds to the intuitions of the user</u>. The most obvious recommendation is to choose a calculator that works the way that the user thinks a calculator should work, i.e., match the characteristics of the machine to the intuitions of the user. In our study of 33 novice and 33 experts, we found that Texas Instruments calculators gave answers that were

most consistent with answers given by our subjects. However, there were disagreement between our subjects' answers and TI's answers on about 20% of the problems for both experts and novices. Rockwell and Sharp gave even poorer matches to our subjects' performance on the problems we used [see 8]. Far more study is required using more problems, different types of calculators, and more subjects, before any definitive conclusions can be made concerning which calculators have the most intuitive operating system. Thus, it is beyond the scope of this paper to provide endorsements for specific calculators. However, the present study suggests that we cannot rely on choosing an "intuitive" calculator as the solution to all our problems, because even the best fitting calculator (i.e., in this case, TI) is considerably different in performance from what our subjects expect. Thus, there is need for instruction that helps produce user conceptions that are more consistent with the way calculators work.

(2) Instruct users in the concepts underlying calculator language. In particular, users should be able to relate each botton push to a series of transactions, i.e., to a description of events taking place in the display and registers. The transaction approach advocated earlier [6] for BASIC appears to apply equally well to calculator language. The locations should be made explicit and visible to the learner, perhaps by providing an erasable "scoreboard" for the display and internal registers. For each press, the learner should be able to alter the contents of the internal registers and display in accordance with the actual transactions.

3. Provide diagnostic tests and remediation based on the user's underlying concepts. The present study has shown that although two users may be equally proficient at using their calculators to solve standard arithmetic

problems, they may differ greatly in their conception of the calculator's operating system. Thus, a purely performance-based test of calculator skill does not tell a teacher what the user "knows". Instead, diagnostic tests should be carried out at the transaction level, e.g., asking the user to state what operations are applied to what objects at which locations in the calculator for each key press. Then, remediation can be provided at the transaction level. For example, if a user indicates that the register is cleared to zero for any non-standard sequence, this can be corrected by showing exactly what happens at a transaction level. Again, a concrete model (such as erasable display and registers scoreboard) could be used.

4. <u>Challenge users to develop procedures for complex problems</u>. Once users have mastered the basics of calculator language, and have developed appropriate conceptions of the underlying transactions, students should be encouraged to transfer their knowledge to more challenging problems. For example, a student who understands the transactions involved in "incrementing display" could be asked to figure a way to multiply a series of numbers, each by the same constant. Or, the user could be given a problem that involves a geometric progression, etc. Many exercise books are available [10, 14], but few provide any training on the principles underlying successful performance on creative problems.

5. <u>Build on calculator language as a means to teach other languages such as BASIC</u>. Students have intuitions about how calculators work. This study has shown that the intuitions of any individual user are fairly consistent. A teacher can build on these intuitions, and use them for transfer to programmable calculators, to BASIC and other languages.

6. Use the calculator as a starting point for the development of computer literacy. Through interactions with calculators the user may develop either a black box or a glass box approach to computers. It is important to start early in helping children (and adults) see that calculators can be understood, for such an attitude is likely to transfer to other human/machine interactions. Mastery of the concepts underlying calculators is just a first step down the road to computer literacy.

Footnotes

This work was supported by Grant NIE-G-80-0118 from the National Institute of Education, Program in Teaching and Learning. Requests for reprints should be sent to: Richard E. Mayer, Department of Psychology, University of California, Santa Barbara, CA 93106.

1. We use the term "operating system" in the most general sense to refer to a program that establishes the mode of user-machine interaction and provides for efficient control of system components. Thus, in the present article, the terms "control program" or "instruction set" could be substituted for "operating system". The way we use the term in this article does not fit the strict definition of "operating system", i.e. a system for mediating among the demands of multiple users in a time sharing system.

2. The main difference between experts and novices is that all of the experts had formal instruction in computer programming and had some introduction to operating systems while none of the novices did. As might be expected, there were other demographic differences between the groups: experts were older, $t(60) = 2.66$, $p < .01$, and experts scored higher in SAT-Mathematics, t ([illegible]4) $= 4.67$, $p < .001$. Thus, while the main comparison was between "liberal arts" students who had no formal programming experience and "engineering" students who had formal training in programming, any comparisons between the two groups must be made in light of other group differences such as age and SAT scores. Individual analysis of the performance of novices who scored high in SAT revealed that they did not perform any more like the experts as compared to the novices who scored low in SAT.

3. The categorization of subjects as indicated in Tables 3 through 7, was based on an analysis of all the problems (i.e, 88 responses) rather than

just the few examples given in the text of this report. Subjects were classified using a forced choice procedure, so that each subject was placed into the category that was most consistent with data that he or she provided us with. Chi square tests were conducted on the data in each of the Tables 3 through 7, using a 2 x 2 contingency table and Yates corrective. The expected frequencies were based on the overall mean for each category, and tested the null hypothesis that there was no difference between experts and novices in the pattern of category frequencies.

4. For example, the most frequently owned calculators were Texas Instruments, Rockwell, and Sharp. The answers given by each of these calculators for each of the 88 problems was compared to the answers given by each subject. Difference scores were computed by counting the number of times that the subject gave an answer that was different from a given brand. For novices who owned TI calculators the difference score was the lowest for TI (8.0) and the highest for the Rockwell (20.0), Sharp (14.1) models being in between. However, for students who owned calculators other than TIs the same pattern was obtained with the lowest difference score for TI (9.8) and higher scores for Rockwell (21.6) and Sharp (15.8). A similar pattern was obtained among experts: for TI owners and non-owners their predictions most closely corresponded to the performance of a TI calculator rather than other brands. Analyses of variance indicated no differences between TI-owners and owners of other brands.

5. There are 13, rather than 16, productions because productions P1, P3, and P9 were never incorporated to the 88 test problems.

6. Tables 8 and 9 describe production systems for actual individual subject rather than composite.

References

1. DuBoulay, B. & O'Shea, T. How to work the LOGO machine. Edinburgh: Department of Artificial Intelligence, Paper No. 4, 1976.

2. DuBoulay, B., & O'Shea, T. & Monk, J. The black box inside the glass box: Presenting computing concepts to novices. Edinburgh: Department of Artificial Intelligence, Paper No. 133, 1980.

3. Greeno, J. G. Cognitive objectives of instruction: Theory of knowledge for solving problems and answering questions. In Cognition and Instruction, D. Klahr, E., Erlbaum, Hillsdale, N.J., 1976.

4. Mayer, R. E. Different problem solving competencies established in learning computer programming with and without meaningful models. Journal of Educational Psychology, 1975, 67, 725-734.

5. Mayer, R. E., Some conditions of meaningful learning of computer programming: Advance organizers and subject control of frame sequencing. Journal of Educational Psychology, 1976, 68, 143-150.

6. Mayer, R. E. A psychology of learning BASIC. Communications of the ACM, 1979, 22, 589-593.

7. Mayer, R. E. Elaboration techniques and advance organizers that affect technical learning. Journal of Educational Psychology, 1980, 72, in press.

8. Mayer, R. E. & Bayman, P. Analysis of user's intuitions concerning the operation of electronic calculators. Santa Barbara: Department of Psychology, Series in Learning & Cognition, Report No. 80-4, 1980.

9. Mayer, R. E. & Bromage, B. Different recall protocols for technical text due to sequencing of advance organizers. Journal of Educational Psychology, 1980, 72, 209-225.

10. Mulish, H. The complete pocket calculator handbook. New York: Collier, 1976.

11. National Council of Teachers of Mathematics. An Agenda for Action: Recommendations for school Mathematics of the 1980s. NCTM, Reston, VA, 1980.

12. Roberts, D. M. The impact of electronic calculators on educational performance. Review of Educational Research. 1980, 50, 71-98.

13. Scandura, A. M., Lowerre, G. F., Veneski, J., & Scandura, J. M. Using electronic calculators with elementary children. Educational Technology, 1976, 16, No. 8, 14-18.

14. Suydam, M. N. State of the art review on calculators: Their use in education. Columbus, Ohio: Calculator Information Center, Report No. 3, 1980.

15. Wertheimer, M. Productive Thinking. Harper & Row, New York, 1959.

16. Young, R. M. The machine inside the machine: Implied models of pocket calculators. Journal of Man-Machine Studies, in press.

Table 1

Sixteen Elementary Calculator Commands

Name	Command	Example	Description
P1	# after #	2 3	Pressing a number key after pressing a number key.
P2	# after +	+ 3	Pressing a number key after pressing a plus key.
P3	# after =	= 3	Pressing a number key after pressing an equals key.
P4	+ after #	2 +	Pressing a plus key after pressing a number key.
P5	+ after +	+ +	Pressing a plus key after pressing a plus key.
P6	+ after =	= +	Pressing a plus key after pressing an equals key.
P7	= after #	3 =	Pressing an equals key after pressing a number key.
P8	= after +	+ =	Pressing an equals key after pressing a plus key.
P9	= after =	= =	Pressing an equals key after pressing an equals key.
P10	# after x	x 3	Pressing a number key after pressing a times key.
P11	x after #	2 x	Pressing a times key after pressing a number key.
P12	= after x	x =	Pressing an equals key after pressing a times key.
P13	x after =	= x	Pressing a times key after pressing an equals key.
P14	x after x	x x	Pressing a times key after pressing a times key.
P15	+ after x	x +	Pressing a plus key after pressing a times key.
P16	x after +	+ x	Pressing a times key after pressing a plus key.

Table 2

Some Possible Transactions in Computer Language

Transaction	Location	Object	Operation	Description
D = D	display	number	no change	No change in the display.
D = 3	display	number	find	Find the old number in the display.
	display	number	destroy	Erase it.
	keyboard	number	find	Find the number that has been entered in the keyboard.
	display	number	create	Put new number in display.
D = R	display	number	find	Find the old number in the display.
	display	number	destroy	Erase it.
	register	number	find	Find the number currently in the register (but do not destroy it).
	display	number	create	Copy the number from the register into the display.
D = eval (R)	display	number	find	Find the old number in the display.
	display	number	destroy	Erase it.
	register	expression	find	Find the expression in the register.
	register	expression	evaluate	Evaluate the expression in the register (but do not destroy it).
	display	number	create	Put the evaluated value of the register in the display.

Table 2 (Continued)

Some Possible Transactions in Computer Language

Transaction	Location	Object	Operation	Description
D = eval (D + R)	display	number	find	Find the number currently in the display
	register	number	find	Find the number currently in the register (but do not alter it).
	register	expression	evaluate	Evaluate the value of the display plus the register.
	display	number	create	Put the new sum in the display.
R = R	register	expression	no change	No change in the register.
R = #	register	expression	find	Find the old expression in the register.
	register	expression	destroy	Erase the old expression from the register.
	keyboard	number	find	Find the number that has been entered in the keyboard.
	register	number	create	Put the new number from the keyboard in the register.
R = "R +"	register	expression	find	Retain the existing expression that is in the register.
	register	operator	create	Place a plus sign to the right of the expression in the register.
R = "R + #"	register	expression	find	Retain the existing expression that is in the register.
	keyboard	number	find	Find the number that has just been entered in the keyboard.
	register	number	create	Place the number to the right of the expression in in the register.

Table 2 (Continued)

Some Possible Transactions in Computer Language

Transaction	Location	Object	Operation	Description
R = eval (R)	register	expression	find	Find the current expression or number in the register.
	register	expression	evaluate	Evaluate that expression or number.
	register	expression	destroy	Erase the expression from the register.
	register	number	create	Replace it with the evaluation of the old number or expression.
R = eval (D + R)	display	number	find	Find the number currently in the display.
	register	number	find	Find the number currently in the register.
	register	expression	evaluate	Add them together.
	register	number	create	Replace it with the sum.
R = eval (R + R)	register	number	find	Find the number in the register.
	register	expression	evaluate	Add the number to itself.
	register	number	destroy	Erase the old number from the register.
	register	number	create	Replace it with the new sum.
R = 0	register	expression	find	Find the existing number or expression in the register.
	register	expression	destroy	Erase that expression or number.
	register	number	create	Replace it with zero.

Table 3

Three Major Conceptions of When to Evaluate An Expression

Conception	Example: Problem	Example: Answer	Proportion of Subjects: Novices	Proportion of Subjects: Experts
Evaluate as soon as a number key is pressed	2 + 3	5		
	2 + 3 +	5	.21	.06
	2 + 3 =	5		
Evaluate as soon as an operation key is pressed	2 + 3	3		
	2 + 3 +	5	.39	.73
	2 + 3 =	5		
Evaluate as soon as an equals key is pressed.	2 + 3	3		
	2 + 3 +	3	.39	.21
	2 + 3 =	5		

Note. - For 2 x 2 contingency table, $\chi^2 = 7.44$, df = 1, $p < .01$.

Table 4

Five Conceptions of How to Evaluate an Arithmetic Chain

Conception	Example: Problem	Example: Answer	Proportion of Subjects: Novices	Proportion of Subjects: Experts
Evaluate in order from left to right.	2+3x7=	35	.88	.70
	2x3+7=	13		
Evaluate backwards from right to left.	2+3x7=	23	.03	.00
	2x3+7=	20		
Evaluate only the last computation.	2+3x7=	21	.03	.00
	2x3+7=	10		
Evaluate multiplication before addiiton.	2+3x7=	23	.03	.30
	2x3+7=	13		
Evaluate addition before multiplication.	2+3x7=	35	.03	.00
	2x3+7=	20		

Note. - For 2 x 2 contingency table, $\chi^2 = 6.98$, df = 1, $p < .01$.

Table 5

Three Major Conceptions of How to Evaluate Equals After Operator

	Example		Proportion of Subjects	
Conception	Problem	Answer	Novices	Experts
Ignore the non-standard sequence.	7+=	7	.82	.76
	7x=	7		
Reset the display.	7+=	0	.09	.06
	7x=	0		
Increment the display.	7+=	14	.09	.18
	7x=	49		

Note. - For 2 x 2 contingency table, $\chi^2 = .52$, df = 1, p = n.s.

Table 6

Three Major Conceptions of How to Evaluate Two Consecutive Operators

Conception	Example		Proportion of Subjects	
	Problem	Answer	Novices	Experts
Ignore the non-standard sequence.	2++=	2	.85	.73
	2xx=	2		
Reset the display.	2++=	0	.03	.00
	2xx=	0		
Increment the display.	2++=	6 or 8	.12	.27
	2xx=	8 or 16		

Note. - For 2 x 2 contingency table, $\chi^2 = 1.53$, df = 1, p = n.s.

Table 7

Three Major Conceptions of How to Evaluate Operation Following Equals

Conception	Example		Proportion of Subjects	
	Problem	Answer	Novices	Experts
Ignore the non-standard sequence.	2x=x	2	.85	.82
Reset the display.	2x=x	0	.15	.00
Increment the display.	2x=x	4 or 8 or 16	.00	.18

Note. - For 2 x 2 contingency table, $\chi^2 = 4.58$, df = 1. $p < .05$.

Table 8

Production System for Subject N

Production Number	Condition	Action	Description
P2	If # after +	then Set D=#, Set R="R+#"	Delayed evaluation and display.
P4	If + after #	then Set D=D, Set R=eval (R)+	Delayed display and immediately evaluated register.
P5	If + after +	then Set D=D, Set R+=R+	No change in display or register.
P6	If + after =	then Set D=D, Set R=R+	No change in display, plus added to register.
P7	If = after #	then Set D=eval (R), Set R=eval (R)	Immediate evaluation and display.
P8	If = after +	then Set D=eval (R), Set R=eval (R)	Immediate evaluation and display.
P10	If # after x	then Set D=#, R = "R*#"	Delayed evaluation and display.
P11	If x after #	then Set D=D, Set R=eval (R)*	Delayed display and immediately evaluated register.
P12	If = after x	then Set D=eval (R), Set R=eval (R)	Immediate evaluation and display.
P13	If x after =	then Set D=D, Set R=R*	Delayed evaluation and display.
P14	If x after x	then Set D=D, Set R=R*	No change in display or register.
P15	If + after x	then Set D=D, Set R*=R+	Set register sign to add.
P16	If x after +	then set D=D, Set R+=R*	Set register sign to multiply.

NOTE.-- Subject evaluates for equals sign only; subject ignores non-standard sequences. Quote marks on the right side of an equality means that the entire expression is held in the register; eval (R) on the right side of an equals means that a single value is substituted for the expression previously in the register.

Table 9

Production System for Subject E

Production Number	Condition	Action	Description
P2	If # after +	then Set D=#, Set R="R+#"	Same as subject N.
P4	If + after #	then Set D=eval (R), Set R=eval (R)	Immediate incrementing display.
P5	If + after +	then Set D=eval (D+R), Set R=R	Immediate incrementing display.
P6	If + after =	then Set D=D, Set R=R+	Same as subject N.
P7	If = after #	then Set D=eval (R), Set R=eval (R)	Same as subject N.
P8	If = after +	then Set D=eval (D+R), Set R=eval (D+R)	Immediate incrementing display and register.
P10	If # after x	then Set D=#, Set R="R*#"	Same as Subject N.
P11	If x after #	then Set D=eval (R), Set R=eval (R)*	Immediate evaluation and display.
P12	If = after x	then Set D=eval (D*R), Set R=eval (D*R)	Immediate incrementing display and register.
P13	If x after =	then Set D=eval (D*R), Set R=eval (D*R)	Immediate incrementing display and register.
P14	If x after x	then Set D=eval (D*R), Set R=eval (D*R)*	Immediate incrementing display and register.
P15	If + after x	then Set D=D, Set R+=R*	Same as subject N.
P16	If x after +	then Set D=D, Set R*=R+	Same as subject N.

NOTE.--Subject evaluates for operation or equals sign; subject increments for some non-standard sequences.

CHAPTER 3

DIAGNOSIS AND REMEDIATION OF BUGS IN BASIC

Note

The material in this chapter was published as the following article:

Mayer, R. E., & Bayman, P. Users' misconceptions of BASIC computer programming statements. Communications of the Association for Computing Machinery, in press.

Abstract

In the process of learning a computer language, beginning programmers may develop mental models for the language. A mental model refers to the user's conception of the "invisible" information processing that occurs inside the computer between input and output. In this study, thirty undergraduates learned BASIC through a self-paced, mastery manual and simultaneously had hands-on access to an Apple II computer. After instruction, the students were tested on their mental models for the execution of each of nine BASIC statements. The results show that beginning programmers--although able to perform adequately on mastery tests in program generation--possessed a wide range of misconceptions concerning the statements they had learned. For example, the majority of the beginning programmers had either incorrect conceptions for or no conceptions of statements such as INPUT A, READ A, and PRINT C. This paper presents a catalogue of beginning programmers' conceptions of "what goes on inside the computer" for each of nine BASIC statements.

Key Words and Phrase: programming, BASIC, novices, man-machine interface.

CR Categories: 1.5, 3.66, 4.2

I. Introduction

The main focus of this paper concerns "what is learned" when a beginning user is taught a computer programming language such as BASIC. The outcome of learning can be viewed in two distinct ways: (1) Learning BASIC involves the acquisition of new information and new rules, such as when to use quotes in a PRINT statement or how to produce a conditional loop using an IF statement. (2) Learning BASIC involves the acquisition of a mental model, such as the idea of memory spaces for holding numbers.

The present study explores the idea that learning of BASIC involves more than the acquisition of specific facts, rules, and skills. Beginning programmers also develop mental models for the language in the process of learning the essentials of BASIC. Moran (6) suggests that the user develops a "conceptual model" of the system as he or she learns to use it. Moran defines the user's conceptual model as the knowledge that organizes how the system works. Users' models may not be accurate or useful representations of "what is going on inside the computer." However, most instructional effort is directed solely at helping the learner acquire the new information and behaviors without giving much guidance to the learner for the acquisition of useful mental models.

Mayer (3) has suggested a framework for describing the internal transformations that occur for elementary BASIC statements. In particular, any BASIC statement can be conceptualized as a list of transactions. A transaction is a simple proposition asserting some action performed on some object at some location in the computer. For example, LET D = 0 involves the following transactions: Find the number is memory space A. Erase the number in memory space A. Find the number indicated on the right of the equals sign. Write this number in memory space A. Find the next statement in the program.

Experts and novices are likely to differ with respect to their mental models for programming statements. For example, an expert programmer may have developed an accurate conception for a counter set LET, such as the one given above. However, the novice may lack a coherent mental model or may possess incorrect conceptions for BASIC statements. In a recent study, Mayer & Bayman (5) found that novice and expert calculator users differed greatly in their conceptions of "what goes on inside the calculator" for various key presses.

How much training one needs to acquire a conceptual model (or mental model) has not yet been explored in research. A first step in addressing this issue is to study novice users' understanding of newly learned programming statements. In this paper, we describe novice users' conception of nine BASIC statements, using a transactional analysis.

II. Method

The subjects were 30 college undergraduates who had no prior experience with computers or computer programming. The subjects took a modified version of a self-instruction, self-paced, mastery course called BASIC in Six Hours (2) that is widely used for teaching BASIC in the Microcomputer Laboratory of the University of California, Santa Barbara. The instruction occurred over the course of three sessions and involved both a programmed manual and hands-on access to an Apple-II computer. Subjects were required to pass mastery tests over one section before moving on to the next section of the manual.

Following successful completion of the course, the users were given a procedure specification test. For each of nine statements (from the instructional course) subjects were asked to write, in plain English, the steps that the computer would carry out for each statement. Users were instructed to write each step on a separate line of the test sheet. The method also involved additional tests which are described in a more detailed report of this study (1).

III. Results

A. Scoring

Each user's protocol for each of the nine statements was broken down into a list of transactions by two scorers, as described in an earlier paper (3).

Three types of transactions were observed for each statement: (1) correct transactions--For example, the key correct transaction for LET A = B + 1 is "store the value of B plus 1 in memory space A". (2) incomplete transactions--For example, a user's answer for LET A = B + 1 could include "store the value of B + 1 in memory." (3) incorrect transaction--For example, a user's answer for LET A = B + 1 could include "store the equation A = B + 1 in memory."

B. Differences among statements.

In order to make comparisons of users' conceptions among the nine statements, each user's verbal protocol for each statement was categorized as correct, incomplete, empty, or incorrect. The criteria for classifying protocols were: correct--if the user's protocol included the key correct transaction(s) and no incorrect transactions, incomplete--if the user produced incomplete versions of the key transaction(s) and no incorrect transaction, and empty--if the subject produced no correct or incomplete version of the key transaction(s) and no incorrect transaction, and incorrect--if the subject produced one or more incorrect transactions. Table I presents a summary of the percentage of users who produced each type of conception for each of the nine BASIC statements. Table I presents the nine statements in order of difficulty based on proportion correct conceptions.

Insert Table I about here

C. Frequency of Misconceptions

For each of the nine statements, a frequency table was generated by tallying the number of occurrences of each correct, incomplete, and incorrect transaction in the protocols of the 30 users. Table II lists the key correct transaction(s) and the three most common incorrect or incomplete transactions (i.e., misconceptions) for each of the nine statements.

Insert Table II about here

IV. Summary and Recommendations

Users' lack of understanding and misconceptions can be summarized as follows:

1. INPUT Statements. Users have difficulty in conceiving where the to-be-input data comes from (i.e., the keyboard) and how it is stored in memory (i.e., in the indicated memory space). Furthermore, many users fail to understand the nature of executive control--i.e., that the computer will "wait" for input from the keyboard as cued by the question mark on the screen. A major misconception is that "INPUT A" means that the letter A is input and stored in memory. These users need explicit training concerning the role of the input terminal, the wait-run control, and the memory spaces.

2. READ-DATA Statements. Users have difficulty in conceiving of where the to-be-read data comes from (i.e., the input queue or DATA statement) and how it is stored in memory (i.e., in a specified memory space). For example, a major misconception is that "20 DATA 80, 90, 99" means that the numbers are

placed into memory or printed on the screen; another major misconception is that "30 READ A" means to find the value of A and print that value on the screen. Subjects need explicit training concerning the data stack and memory spaces.

3. Conditional and Simple GOTO Statements. Users' major difficulty with the GOTO statement is that they do not understand what will happen next, after program execution moves on to the desired line. Also, with the conditional GOTO, users have difficulty in understanding what to do if the condition is false. Misconceptions include thinking that "60 GOTO 30" means to find the number 30 (rather than line 30) and "IF A < B GOTO 99" means move to line 99 (without a test). Hence, beginners need training in executive control of the order of execution of statements in a program.

4. LET Statements. Users seem to get confused between solving or storing an equation (i.e., treating the equal sign as an equality) and making an assignment. Those who seem to understand the assignment property in the statement still have difficulties in conceiving where to store the assigned value. For example, a major misconception for "LET A = B + 1" or "LET D = 0" is to think that the equation is stored in memory. Beginning programmers need explicit training concerning memory locations and under what conditions values stored in those locations get replaced.

5. PRINT Statements. Users seem to confuse the function of PRINT C and PRINT "C". Also, users have difficulty in conceiving that these statements simply display on the screen what is asked to be printed; users incorrectly assume that the computer keeps a record of what is printed somewhere in memory. Beginning programmers need explicit training in comparing the types of PRINT statements.

The present study provides evidence that "hands-on experience" is not sufficient for the productive learning of computer programming by novices. Users tend to develop conceptions of the statements that either fail to include the main idea or that include outright misconceptions. Explicit training is needed including the introduction of a concrete model showing the key locations in the computer (e.g., memory spaces, input stack, etc.), verbal and visual descriptions of the key transactions for each statement, and encouragement of the user to role play "what the computer is doing" for statements and programs. These techniques are reviewed elsewhere (4).

References

1. Bayman, P. & Mayer, R. E. Novice users' misconceptions of BASIC programming statements. Dept. Psych., Series in Learning and Cognition, Report No. 82-1, Santa Barbara, 1982.

2. Marcus, J. Basic in Six Hours: A Self-Instruction Text. Santa Barbara, CA: Microcomputer Laboratory, 1980.

3. Mayer, R. E. A psychology of learning BASIC. Comm. ACM 22, 11 (Nov. 1979), 589-593.

4. Mayer, R. E. The psychology of how novices learn computer programming. Comput. Surv. 13, 1 (March 1981), 121-141.

5. Mayer, R. E., and Bayman, P. Psychology of calculator languages: A framework for describing differences in users' knowledge. Comm. ACM 24, 8 (Aug. 1981), 511-520.

6. Moran, T. P. An applied psychology of the user. Comput. Surv. 13, 1 (March 1981), 1-11.

Table I

Percentage of Users with Correct, Incomplete, Incorrect, and Empty Conceptions for the Nine BASIC Statements

	Conceptions			
Statements	Correct	Incomplete	Incorrect	Empty
INPUT A	3%	30%	30%	37%
30 READ A	10%	27%	17%	47%
IF A < B GOTO 99	27%	27%	40%	7%
LET A = B + 1	27%	10%	60%	3%
20 DATA 80, 90, 99	27%	17%	13%	43%
60 GOTO 30	27%	56%	10%	7%
PRINT C	33%	0%	47%	20%
LET D = 0	43%	3%	53%	0%
PRINT "C"	80%	0%	13%	7%

Table II

Percentage of Users Who Produced Key Correct and Incorrect or Incomplete Transactions for Nine BASIC Statements

Transaction	Percentage of Users
INPUT A	
*Print ? on the screen.	7%
* Wait for a number and <CR> to be entered	23%
*Write the entered number in memory space A.	3%
Write A in memory (or a data list).	30%
Wait for a certain number or letter.	13%
Print A on the screen.	3%
30 READ A	
*Write next number from DATA in memory space A.	10%
Print value of A on the screen.	10%
Write letter A in memory.	3%
Wait for number to be entered from keyboard	3%
IF A < B GOTO 99	
*If the value of A is less than B, then move to line 99 in the program.	63%
*If the value of A is more than or equal to B, then move on to the next statement in the program.	33%
Move to line 99 (without test).	10%
Print the number 99 or line 99 on the screen.	10%
Write A or B or A<B or a number in memory.	10%

TABLE II (continued)

Transaction	Percentage of Users
LET A = B + 1	
*Write the obtained value in memory space A.	30%
Write the equation in memory.	43%
Write B + 1 in memory space A.	33%
Print A=B+1 or A or the value of A on the screen.	23%
20 DATA 80, 90, 99	
*Put numbers in input/queue.	27%
Put numbers in memory.	20%
Print numbers on screen.	13%
Put numbers in memory space A.	7%
60 GOTO 30	
*Move to line 30 in the program.	67%
*Continue from there.	37%
Find the number 30.	10%
Move to line 30 if A does not equal a certain value.	7%
Print line 30 on screen.	3%
PRINT C	
*Print number (or 0) on screen.	40%
Print the letter C on the screen.	33%
Write the letter C in memory.	7%
Print either "error" or nothing on the screen.	7%

TABLE II (continued)

Transaction	Percentages of Users
LET D = 0	
*Write zero in memory space D.	47%
Write the equation in memory.	47%
Write D or 0 in memory.	13%
Print the equation on the screen.	7%
PRINT "C"	
*Print the letter C on the screen.	83%
Write the letter C in memory.	7%
Print the value of C on the screen.	7%
Find the number in memory space C.	3%

Note.--Asterisk (*) indicates key correct transaction for each statement. The three most common incorrect or incomplete transactions are also listed for each statement.