ED 296 703 IR 013 330

AUTHOR Schwartz, Steven; And Others
TITLE An Empirical Study of a "Metacourse" To Enhance the Learning of BASIC. Technical Report.
INSTITUTION Educational Technology Center, Cambridge, MA.
SPONS AGENCY Office of Educational Research and Improvement (ED), Washington, DC.
REPORT NO ETC-TR87-7
PUB DATE Sep 87
CONTRACT 400-83-0041
NOTE 60p.
PUB TYPE Reports - Research/Technical (143) -- Tests/Evaluation Instruments (160)

EDRS PRICE MF01/PC03 Plus Postage.
DESCRIPTORS Cognitive Processes; Error Patterns; High Schools; *Instructional Effectiveness; Intermode Differences; Models; Pretests Posttests; *Programing; Questionnaires; Teacher Attitudes; *Teaching Methods; *Transfer of Training
IDENTIFIERS BASIC Programing Language; *Metacourses

ABSTRACT

This study examined the effectiveness of a metacourse consisting of eight lessons interspersed over a semester-long beginning course in BASIC and aimed at providing mental models, problem-solving strategies, key concepts, and other heuristic structures. The experimental group consisted of 6 teachers who taught 9 classes of a total of 132 high school students; the control group consisted of 9 teachers who taught 13 classes of a total of 239 high school students. Data collection included a student questionnaire on previous experience with computers, a pre/posttest to assess general cognitive skills at the beginning of the term and possible transfer effects at the end, classroom observations, an end-of-semester test on BASIC, and homework assignments. Analysis focused on teachers' fidelity to the metacourse lessons, the impact of instruction on the students' mastery of BASIC, and the transfer of cognitive skills from programming to other domains. Results showed that teachers were faithful to the lessons and found them quite teachable. Experimental group students made significantly fewer errors on the test of BASIC and did significantly better on all major categories of problems. Evidence of transfer that was observed was limited to a particular problem similar to the programming tasks. The appendices include examples from the metacourse manual, the cognitive skills pre/posttest, the BASIC test, the classroom observation sheet, and the student questionnaire. (34 references) (Author/MES)

ED296703

TR87-7

AN EMPIRICAL STUDY OF A "METACOURSE" TO ENHANCE THE LEARNING OF BASIC

Technical Report

September 1987

Educational Technology Center
Harvard Graduate School of Education
337 Gutman Library   Appian Way   Cambridge MA 02138
(617) 495-9373

IR013330

An Empirical Study of A "Metacourse" to Enhance the Learning of BASIC

Technical Report
September 1987

Prepared by:

Steven Schwartz
D.N. Perkins
Greg Estey
John Kruidenier
Rebecca Simmons

Programming

<u>Group Members</u>

Naomi Bolctin
John Burnette
Susan Cohen
Greg Estey
John Kruidenier
Doug McGlathery
David Niguidula
David N. Perkins
Rebecca Simmons
Steve Schwartz
Tara Tuck

Preparation of this report was supported in part by the Office of Educational Research and Improvement (Contract # OERI 400-83-0041). Opinions expressed herein are not necessarily shared by OERI and do not represent Office policy.

ACKNOWLEDGEMENTS

The "Metacourse" discussed here is the product of the Programming Group at the Educational Technology Center, all the members of which are thanked for their many contributions. Ellen Mandinach of the Educational Testing Service suggested certain test items for the cognitive skills test. We would especially like to thank Bill Bower, Julie Hochstadt, Kathy Hollowell, Patricia Jones, Kathy Moore, Nancy Samaria and Paul Shapiro for their contributions both in and out of their classrooms in the development and assessment of the Metacourse materials.

Table of Contents

An Empirical Study of a "Metacourse" to Enhance the Learning of BASIC

Enhancing instruction in an existing subject matter on a wide scale is a conspicuous and rarely met challenge of contemporary education, and for understandable reasons. On the one hand, the subject matters bring with them a number of conceptual challenges and problems of metacognitive control that require careful study to disclose and careful instructional design to remedy. On the other hand, however artfully fashioned better instruction may be, wide-scale implementation must run the gauntlet of a number of impeding factors -- teacher training, cost of materials, institutional inertia, and so on.

A desirable program of research would address notonly the learning of a subject matter but also the practical problems of implementation as part of a unified program of inquiry. During the past several years, we have tried to conduct such an investigation into the pedagogy of programming. Through clinical studies and teaching experiments, we have sought to understand better the factors that interfere with beginners' mastery of programming in BASIC and LOGO and to devise instructional methods that enhance their learning (see e.g. Perkins, Farady, Hancock, Hobbs, Simmons, Tuck, & Villa, 1986; Perkins, Hancock, Hobbs, Martin, & Simmons, 1986; Perkins, Hancock, Hobbs, Martin, & Simmons, 1986; Perkins, Martin, & Farady, 1986). Regarding implementation, we have tried to couch our instructional methods in a format called a "metacourse," designed from the first to address some of the daunting problems of wide-scale disseminability (Perkins, Farady, Simmons, &Villa; Perkins, Schwartz, & Simmons, in press).

This paper describes a large-scale experiment examining the effectiveness of the metacourse we have developed in enhancing high school students' learning of BASIC. The positive results can be taken as support for the general analysis of the difficulties of programming developed in our earlier work, and for the viability of our approach to educational change. Before detailing the experimental method and results, we describe briefly the context established by our prior work.

Students' learning problems in mastering programming

It is plain that computer programming poses special challenges. Like mathematics and physics, programming is a precision-intensive subject matter, requiring meticulous care with details. Programming is also problem-solving intensive, student activities focusing almost entirely on resolving programming problems. While today this is typical of mathematics, physics, and certain other science subjects, we note that any school subject can and perhaps should be treated in a problem-solving intensive way. Finally, programming is not just problem-solving intensive but "design intensive." That is, students have to construct whole complex products that do certain jobs, not just derive particular answers like 35 cm/sec or A=17. In today's schools, complex products appear mostly in art (the works themselves), Euclidean geometry (proofs), and English and Social Studies (essays). Again, we note that any school subject can and perhaps should emphasize the construction of complex products.

It is hardly surprising that a precision-intensive, problem-solving intensive, and design-intensive subject matter should give many students considerable difficulty. In recent years, evidence has accumulated that a high percentage of students in elementary and high school achieve only extremely limited mastery of programming even after a semester or two of instruction (see e.g. Mawby, 1987; Pea, 1986; Pea & Kurland, 1984a; Pea & Kurland, 1984b; Kurland, Pea, Clement, & Mawby, 1986; Kurland, Clement, Mawby, & Pea, 1987; Bonar & Soloway, 1985; Soloway & Ehrlich, 1984; Sleeman, Putnam, Baxter, & Kuspa, 1986). In seeking to summarize and synthesize our own and others' findings in this area, we have found it useful to characterize students' difficulties under three broad headings: fragile knowledge, a shortfall in elementary problem solving strategies, and problems of confidence and control.

<u>Fragile knowledge</u>. "Fragile knowledge" refers to the fact that students commonly display partial knowledge, considerable inert knowledge (not evoked in contexts of need but retrievable with cueing), and garbled knowledge (concepts used in the wrong place, in inappropriate hybrids) of programming. Perkins, Hancock, Hobbs, Martin, and Simmons (1986) discuss fragile programming knowledge in detail, emphasizing the importance of distinguishing between fragile and missing knowledge: Students typically have much more knowledge than they use well. If students could somehow activate their inert knowledge and perform internal cross-checks of garbled knowledge, they might perform substantially better.

<u>Elementary problem-solving strategies</u>. By asking themselves elementary "problem management" questions like "What am I trying to do now," "do I know a command that could help," "exactly what does the line of code I just wrote do if I hand execute it," and so on, students might make better use of their fragile knowledge base. Clinical experiments reported in Perkins, Hancock, Hobbs, Martin, and Simmons (1986) and Perkins, Martin, and Farady (1986) suggest that this is so. Unfortunately, students do not appear to probe with such questions as often as they might, a shortfall in elementary problem-solving strategies.

<u>Confidence and control</u>. Finally, students often evince motivational problems that interfere with their controlling well their own problem-solving processes. For example, many students simply disengage from programming problems and commence a side activity or seek help as soon as the least difficulties emerge. These "stoppers" as we have called them are often quite capable of continuing on non-directive prompting, but do not seem to recognize their own abilities (Perkins, Hancock, Hobbs, Martin, & Simmons, 1986).

One way or another, efforts to enhance instruction in programming should address the triple problem of fragile knowledge, strategic shortfall, and confidence. Note that these difficulties as described here do not force wholesale reconsideration of what is taught in elementary programming instruction. After all, the key commands, the fundamental operating procedures, and so on, surely need to be taught. Instead, the shortfalls identified could be taken to invite some sort of "booster shot" -- some treatment to enhance the learning set and strategic repertoire students bring to the enterprise of programming.

The notion of a metacourse

With this point in mind, we introduce the notion of a "metacourse." The goal of a metacourse is to provide mental models, problem-solving strategies, key concepts, and other structures that may help students to understand more deeply and wield more artfully the knowledge they are acquiring during their regular instruction in a subject matter. In particular, the Metacourse in Programming discussed here offers students a mental model of the computer and how it works at a level appropriate for understanding BASIC and its operation, a strategy for understanding new commands and relating them to the mental model of the computer, several strategies for breaking programming problems down into subproblems of various sorts, and other concepts and tactics designed to help students deal with the difficulties identified earlier (Perkins, Schwartz, & Simmons, in press).

The metacourse is not a course -- a complete remaking of the usual instruction. Rather it functions in a "meta" way, infusing a few important and often neglected conceptual elements into "business as usual." Thus, the Programming Metacourse is organized to allow the teacher to introduce key concepts periodically as the term advances and students gain a knowledge base in BASIC. This infusion process, together with the metacognitive emphasis, are the factors that distinguish metacourse design from conventional curriculum redesign.

Why might a metacourse offer an approach to enhancing instruction that evades some of the implementation difficulties outlined earlier? In comparison with a new course altogether, a metacourse is much more compact and its materials much less expensive. A metacourse does not displace, but merges smoothly with, existing materials and instructional practices. Accordingly, we suggest that a metacourse lends itself to wide-scale dissemination easily more than most efforts at curriculum reform.

The results of a preliminary study reported in Perkins,Farady, Simmons and Villa (1986) indicated that the Metacourse was eminently teachable, with no major problems of teacher preparation. While there was some encouraging evidence that the Metacourse was having its intended effect, the data from this formative study were far from conclusive.

In the following pages, we report on a large scale experiment involving a number of treatment and control classroom attempts to provide further evidence on the effectiveness of the Metacourse in settings where emphasis is placed on infusing the key concepts into the teacher's presentations throughout the entire course.

Method

Setting

The Programming Group's second empirical study of the Metacourse was conducted within the context of another Educational Technology Center study, the "Laboratory Sites" project. This broader study, described in detail in another report, aimed to involve the teachers using the Metacourse (as well as interventions in two other subject matter areas) in the discussion and evaluation of the educational innovations they were implementing in their classrooms.

Participation in the Laboratory Sites project meant that the teachers using the Metacourse received more support in their efforts to adapt the new materials to their own styles and classrooms than would normally be the case. The main extra supports were: monthly meetings with the research team to discuss issues pertaining to the implementation of the Metacourse; frequent contact with a research assistant or the Laboratory Sites project liaison assigned to the Programming Group; access to all Educational Technology Center personnel, as well as each other, via a personal computer communications network and a small stipend in compensation for their participation in collaborative research.

Although not all of these supports may have been significant for all of the teachers in the experimental group, taken together they constitute an enriched support environment not normally present, and not present in our control sites. The control site teachers, however, were not faced with the problem of introducing and infusing new, innovative materials into their well practiced normal BASIC curriculum.

From the standpoint of the research, the participation of the Programming Group in the Laboratory Sites project created opportunities for very precise and timely feedback from teachers about the effectiveness of the different lessons in the Metacourse. This feedback, which would not otherwise have been available proved to be extremely valuable in guiding further revisions to the Metacourse.

Subjects

The experimental group consisted of 6 teachers of BASIC, who taught 9 classes and 132 students at 5 high school laboratory sites. The control group pool consisted of 9 teachers who taught 13 classes and 239 students at 8 control sites. The large control group was used so that the groups could be matched if necessary on the basis of the general cognitive pre-test if overall pre-test scores indicated that one group or the other was significantly more able. The difficulties in data analysis encountered in the first empirical study, in which the control group proved much more able on the general cognitive measures, might thus be avoided.

All teachers were experienced programming instructors, beginning the semester with at least 2 years of previous experience teaching BASIC classes. All of the classes were straightforward programming classes, meeting on average 5 times a week for forty-minute periods throughout one semester.

Procedures

Teaching Intervention

All of the experimental group instructors met with the Programming Group staff and research assistants early in the semester in order to go over the Metacourse materials and procedures. Each teacher was introduced to a particular research assistant who would observe two Metacourse lessons and one non-Metacourse lesson.

As described in the introduction, the Metacourse is a series of instructional lessons designed to enhance and supplement the material typically covered in a first-semester BASIC course. One major concern is to provide students with a clear mental model of the computer in the BASIC environment and to help students internalize and employ the model when appropriate. Another goal of the Metacourse is to provide students with programming-specific skills and a conceptual framework that guide the student in initial understanding and subsequent application of material learned in class.

The eight lessons are designed to be interspersed throughout the semester. Preliminary lessons provide students with a visual model of the computer and equip students with the conceptual framework for understanding commands, a framework that involves thinking of a statement in terms of its purpose, syntax, and action (semantics). Later lessons deal with strategies for writing good programs including breaking programs down into manageable chunks, thinking of code in terms of functional units called "patterns", and regular use of checking and debugging strategies. The components of the Metacourse are described in more detail below.

The "paper computer": A visual model of the computer. The concept of a mental model is key in the design of the Metacourse. Recent work in the field of cognition underscores the importance of helping students construct robust models in various domains (cf. Beveridge & Parkins, 1987; Gentner & Stevens, 1983; Johnson- Laird, 1983; Mayer, 1976, 1981). Further, empirical work in the pedagogy of computer programming suggests that a stronger mental model of the computer can result in increased programming performance (DuBoulay, 1986; DuBoulay, O'Shea, & Monk, 1981; Mayer, 1976, 1981,1985).

In the Metacourse, students learn a visual model of what happens inside the computer during program execution to help them interpret exactly what the various BASIC commands do (See Appendix A). The model is designed to help counter some of the problems of fragile knowledge and strategic shortfall we noted in students in our clinical work. Regular use of the model in learning commands and in envisioning the effects of single lines or chunks of code should serve to help build a robust knowledge base of command effects. In addition, the model can provide students with a fairly simple strategic tool for debugging. Using the visual model to imagine precisely the actions of the computer at a trouble spot gives stoppers a tool to help them get moving again and may assist haphazard movers by providing a focus for their activities.

Specifically, the model depicts variables and their values, characters on the screen, and flow of control. The model is functional rather than technical in nature in an effort to promote visibility and simplicity in the model (cf. DuBoulay, O'Shea, & Monk, 1981; Beveridge & Parkins, 1987). Thus, for example, the student thinks of a variable as the name of a box in memory with a number or character string in it. This model is called the "paper computer," because students receive paper forms displaying the visual model on which they hand execute programs. In addition, an on-line version of the paper computer is being designed, a tutorial that presents the same visual model of the computer and steps through a program line by line, making the requisite changes in the computer state as each line is executed.

Purpose-syntax-action: A framework for program constituents. In the Metacourse students also learn an analytical scheme for understanding commands and command lines. The scheme encourages students to consider the command's purpose, its legal syntax, and its action in the computer world as shown on the paper computer. The purpose-syntax-action framework is used to help students recognize, attend to, and organize the important features of a new command when it is first encountered. Additionally, the framework serves as a problem-solving aid when students write programs. In considering how to accomplish a particular programming task students are asked to think carefully about what commands they know whose purposes may accomplish the task at hand. When writing command lines, students are encouraged to attend to the syntax, and to envision the action of the command, that is, the effect of the line on the computer. Students are also encouraged to use a "minimanual", a quick reference guide that includes the key BASIC commands organized according to the purpose, syntax, and action questions, with examples (See Appendix B). As students gain programming experience and become comfortable in using the framework for individual commands, they are introduced to the utility of employing a similar organizational strategy for larger "patterns" of code. (Patterns will be discussed below.)

The choice of terms in the framework is derived in part from the contrast between programming pragmatics, syntax, and semantics, and in part from a model for learning developed by Perkins (Perkins, 1986a,b). Purpose has a major role in the triad as a means to counter some of the problems of fragile knowledge, specifically that of inert knowledge. In earlier clinical studies it was found that often students had a knowledge of relevant command structures, but could not retrieve them, apparently failing to make the connection between what needed to happen in the program and the commands that would serve the purpose (Perkins & Martin, 1986; Perkins, Martin & Farady, 1986). The attention to syntax is based on earlier observations noting the degree to which BASIC and LOGO program performance among novice programmers can be affected by problems of a purely syntactic nature (Perkins, Farady, Hancock, Hobbs, Simmons, Tuck, & Villa, 1986; Perkins, Farady, & Martin, 1986). The emphasis on the action or semantics of a command line stems from the desire to help students construct and utilize a robust knowledge base of command effects as described above in the discussion of visual models.

Patterns: thinking beyond individual commands. The Metacourse emphasizes the importance of organizing programming knowledge not only in terms of individual commands but also in terms of multiple lines of code that work together to accomplish a particular task or subtask. Such recurrent schema are called "patterns," a term roughly synonomous with the "programming plans" described by Soloway and colleagues (Soloway & Ehrlich, 1984; Joni & Soloway, 1986). Patterns provide an intermediate level of analysis between the whole program and individual command lines, and offer a way of helping students to organize and comprehend code used for frequent programming tasks such as counter variables, certain compound conditional branches, and so forth (See Appendix C). The Metacourse stresses the importance of patterns as an intermediate level coding strategy as well as a debugging strategy.

Writing a program: Planning, coding, checking, and debugging. Test results and teacher comments from our previous study indicated that students needed additional instruction in planning and writing whole programs. Thus the revised Metacourse provides students with a procedure for writing programs that emphasizes the importance of planning, checking, and debugging in addition to the actual coding of a program. Another major point of emphasis is that writing programs is a process of refinement, involving multiple rounds of these four activities.

As a first step in top-level planning, students are encouraged to think in terms of "interactions" with the user. The interaction refers to the sequence of computer outputs and user inputs that occurs as a user uses a program. This heuristic offers a concrete beginning point for the novice programmer, who frequently experiences difficulties in moving from a given problem statement to the initial stages of the task decomposition. For many programs, an outline according to the "rounds" of interaction provides a decomposition into subunits that amount to subproblems in the programming task. An initial focus on interactions as the student begins to program can help to avoid the frequent problem of moving from a problem statement directly to a coding phase without sufficient thought and effort devoted to planning.

From this initial breakdown of the problem, students are asked to conceptualize program sections in terms of the patterns that might serve the section purposes. The patterns provide the student with the tools to manage and create code above the level of a single command line. Students are taught the utility of having a repertoire of patterns that are portable across programs. Thus, for example, if a student recognizes the need in a program to implement code that will trap unreasonable inputs from the user, the student can call on a "bulletproofing" pattern to help with that task. This simple pattern includes a conditional branch that rejects inappropriate inputs, prompting the use for another input.

At each level of the planning and coding stages students are encouraged to consider the purpose of chunks of code and the action those chunks actually effect. Students are taught to check their code, mentally simulating the action of the program line by line to catch the "easy" bugs before the program is actually run and tested. The Metacourse also emphasizes that bugs and debugging are an inevitable and integral aspect of program production, not a reflection of poor programming.

<u>Assessment procedures</u>

A pretest in general cognitive skills was administered at the beginning of the term. At the end of the term, the cognitive skills test was readministered to test for possible transfer effects, and a general BASIC skills test was given to test for mastery of BASIC. Both tests are described below.

<u>Cognitive Skills Pretest-Posttest (Appendix D)</u>. A cognitive skills test was developed with two purposes in mind. First, such a measure might be expected to correlate with programming performance (most cognitive skills are interrelated), and thus serve as an indicator of level of general student ability in comparing treatment and control students. In addition, it seemed sensible to probe the possibility of transfer of cognitive skills. In general, findings on transfer from programming have been negative (Kurland, Pea, Clement, & Mawby, 1986; Mawby, in press; Pea & Kurland, 1984a,b: Solomon & Perkins, in press), nor was the Metacourse specifically designed to promote transfer. However, there have been occasional positive findings (Clements, 1985; Clements & Gullo, 1984) and the issue has great currency, warranting an effort to examine the [illegible] of transfer in the present study.

[illegible] nitive skills instrument was designed to test skill in formal syllogistic reasoning, [illegible] reasoning (e.g. "If the day after tomorrow were Thursday, what would the day before yesterday be?"), field-independence and planning (a task of counting the number of triangles in a complex diagram). In addition, the test incorporated an algebraic version of the well-known students-and-professors problem (Clement, Lochhead, & Monk, 1981) where students typically have great difficulty translating a simple algebra word problem into an algebraic equation. Soloway and his colleagues have suggested that computer programming experience may help students deal with this sort of problem more effectively (Ehrlich, Soloway, & Abbot, 1982; Soloway, Lochhead, & Clement, 1982). A further item relating to accuracy and precision of observation and description, required the student to accurately describe a complex geometric figure in order for another student to be able to recognize it amongst a set of similar complex geometric figures.

The cognitive skills test also included a problem of a type quite similar in character to a programming problem involving combining patterns in a program-like way. Students had to generate a description of some events using a restricted language containing the words "repeat" and "decide". (The design of these problems was suggested by Ellen Mandinach of the Educational Testing Service); in prior research, when the rare case of transfer from programming has been found, it has emerged most often on tasks with marked similarity to programming (cf. Kurland, Pea, Clement, & Mawby, 1986; Linn, 1985).

Finally, in addition to the purely "cognitive" aspects of the pretest, we also included a series of questions designed to address a particular attitudinal factor, locus of control (Rotter, 1966). Research in this area (Dweck & Bempechat, 1983; Dweck & Licht, 1980), as well as studies of performance in the programming domain (Perkins, Martin & Farady, 1986; Zelman, 1985), point to the importance of attending to feelings of confidence and control in student performance. Thus we included a 10 question subtest derived from the "attribution of intellectual responsibility scale" (Rotter, 1966), measuring internality/externality of control in positive and negative intellectual situations.

BASIC Test (Appendix E). The BASIC test was a fairly standard paper and pencil programming test, comprising 16 items. The problems were formulated to evaluate certain general programing skills -- the ability to hand-execute, debug, break a problem into subtasks -- as well as to test knowledge about programming commands typically presented in an introductory BASIC course (e.g. PRINT, LET, INPUT, FOR/NEXT, IF-THEN). The test was also designed to include certain language independent bugs identified by Pea (Pea, 1986) .

The first nine problems tested simple production and recognition skills, usually requiring production of only one line of BASIC code using one command statement. The last seven problems included three hand-execution problems, one debugging problem and three longer programming problems requiring somewhat more complex structures such setting up counters, use of for-next loops and the like. The 16 problems are briefly described below.

Simple one line command problems (1 - 9):

1. PRINT-string: The student is asked to write the code that would result in a specific string
   being printed on the screen.
2. PRINT-number: A specific number is to be printed on the screen.

3. INPUT-number: A user is prompted to type in a number.
4. LET-number: Set a variable to a number.
5. LET-expression: Set a variable to the result of an operation.
6. LET-string: Set a variable to a string.
7. IF/THEN: If a given condition holds, print an expression.
8. FOR-loop: fill in blanks to complete a simple For/next loop.
9. GOTO: Fill in a blank to complete a Goto statement.

Hand execution problems (10, 11 and 13):

10. Given a short program, show what will appear on the screen when the program is executed.
11. Garden path: Same as 10 above, except that what will appear on the screen when the program is executed is not what one would expect -- the hand-executor is led down a garden path.
13. Parallelism bug: Same as 10 above, except that the student is given a program in which the individual commands are correct, but are presented in the wrong order. This was designed to test for skill in recognizing what Pea (1985) calls a parallelism bug, the assumption by programmers that the computer is able to execute more than one line of code simultaneously (in which case order would not be important).

Debugging problem (12 -- see also problems 11 and 13):

12. Egocentrism bug: The student is given a program with an INPUT statement missing because they often assume that the computer is aware of what they are thinking

Production problems (14 - 16):

14. FOR-loop: Using a FOR/NEXT loop, write a program that will print the numbers 1 through 10 on the screen.
15. IF-loop: Same as problem 14, except that the loop is created using an IF statement.
16. Complex: The student is asked to write a more complex program that requires an I INPUT statement inside a FOR loop that accumulates a sum.

Classroom Observations

Researchers conducted classroom observations at all treatment sites, systematically recording a number of features of the classroom dynamics and instructional style, as well as items related to the Metacourse treatment itself. Observations included two Metacourse lessons, as well as one regular class, in each case conducted by a single research assistant assigned to observe one or two teachers. Assignments of multiple observers to a class were not made for two reasons. First, we wished to be as non-obstrusive as possible and thus each teacher and class could get comfortable with a single outside observer. Second, since our main comparsons were between Metacourse and non-Metacourse lessons in the same classes, inter-rater reliability was not an issue.

The majority of the items on the observation instrument (see Appendix F) were rated on 7-point Likert scales (where 7 is the highest rating and most desirable score). These included situational factors such as student- teacher and student-student interaction, student responsiveness and engagement, and teacher presentational style. Items relating specifically to the Metacourse included fidelity to the Metacourse and infusion of Metacourse concepts into the Metacourse lessons. In the area of classroom interactions, the scales measured amount and quality of interactions between students and teachers and among students. Two types of student participation were scored, the degree to which students engaged in process- and product-oriented participation. Students (as a group) were also rated on preparedness for the material presented. Teachers were also rated on mode of instruction (e.g., presentational, interactive) and on degree of comfort with the material presented. Observers noted whether the instruction included any bridging activities between programming and other academic disciplines or real-life situations. Finally, observers used the 7-point scale to give an overall rating of effectiveness of instruction.

In the observations of Metacourse lessons, fidelity to the lessons was rated, based on adherence to various aspects of the lesson (introduction, lecture, participation, explanation, and exercises). The Metacourse lessons were also rated according to the degree to which the basic principles of the Metacourse had been integrated into the lesson.

Student Questionnaire

At the beginning of the semester, students filled out a short questionnaire designed to help assess their previous experience with computers and computer programming (See Appendix G). Items were designed to determine the students' general experience with computers in the schools, with computers in the home, and in other outside activities (e.g., computer camps). In addition, students were asked to report previous experience in the BASIC programming language as well as other programming languages.

Homework

Most of the nine Metacourse lessons included worksheet problems for students to complete in class and homework assignments to be completed outside of class between lessons. The task of sending homework and worksheet papers to ETC was rotated among the teachers so that, for each lesson, different teachers were asked to return their students' papers. These were xeroxed and returned as soon as possible to the teachers.
Teachers returned 83% of the sets of student homework and worksheets asked for (25 out of 30). Teacher response was better toward the end of the Semester than the beginning. Each student's homework was "graded" and the total number of students making a particular number of errors on each problem was recorded.

Coding Procedures

Cognitive Pretest-Posttest. The cognitive test included several types of problems. The-days-of-the-week problems and the students-and- professors (formula) problem were scored either 0 for correct,or 1 for incorrect. In the triangle-count problem, the correct answer was scored 0. If the student reported more than the correct number of triangles, the score was the excess with a minus sign; if fewer, the score was the shortfall with a plus sign.

The description-of-geometric-figure problem and the repeats-and-decides problems required special conventions because of the complexity of possible responses. In the geometric problem, the following categories were scored: 1) right figure kept, 2) geometric shape omission, and 3) position omission. If the correct figure was kept the student scored a 0, and one point was added for each shape in excess kept. For the shape category, the student scored a 0 for correct description of shapes, 1 if one shape was omitted, 2 if both shapes (square and rectangle) were omitted. If the correct positioning of the squares was correct the student scored a 0, 1 for one of the relationships was missing, 2 if both relationships (top-bottom/spread out) were omitted.

In the repeats-and-decides problem, form, order and content were defined as scoring categories. Form referred to the writing of commands in the correct format. Order referred to the correct ordering in a sequence of commands. Content concerned whether the student's repeats and decides "program" would do the assigned task. All categories were scored 99 if the student did not try the problem. Otherwise the categories were scored on a 0 to 4, where 0 designated a perfect answer, 1, a single error and so forth up the scale to 4 which indicated insufficient data/ misunderstood task or for the form category, repeats and decides not used.

The Attribution of Intellectual Responsibility scale was scored in the usual manner in terms of which alternative (internal or external locus of control) was chosen on the 5 positive and 5 negative intellectual situations.

BASIC Test. The students' performance on the BASIC test was scored in a number of different categories. Many of the categories only suited certain problems, for instance, the counter category only applied to problems that included a counter. Each of the 16 problems was scored for errors by noting whether or not a student's response included certain specific features. Responses to Problem 1 (PRINT-string), for example, were scored for two features, or error categories. To be error free, a response needed to include a correct PRINT command (the first error category) as well as a quoted string (the second category). The sample response (1) below would be scored as having 0 errors -- it is correct. Response (2), however, would be scored as having 1 error; the PRINT is correct, but the quote category is scored as incorrect because the first quotation mark is missing.

```
(1) PRINT "the answer is cat"_
(2) PRINT  the answer is cat"_
```

Some problems called for an "extraneous statements" category. This was scored when an extra statement or set of statements was added to the program, which may or may not have resulted in an error, but showed clearly that the student had an improper model of machine semantics. Several other categories corresponded to the use of various program constructs; for instance the positioning of PRINT statements, FOR-NEXT statements and the like. The number of error categories scored for a problem ranged from one in problem 9 (the fill-in-the-blank GOTO problem) to ten in problem 16 (the complex production problem).

Scoring Procedure. Using the scoring system described earlier, 4 scorers on the cognitive test and 2 scorers on the BASIC test worked independently to code the students' responses to the test, crosschecking periodically with one another and establishing policies to clarify the scoring system. The coding was blind with respect to whether the response came from the treatment or control group, or from a pre or post-test. While every student response was not coded by all scorers, the scorers all coded and discussed a random subsample. Policies that could affect prior scoring were applied retroactively.

After about ten percent of the data had been scored the scorers deemed themselves to be adequately calibrated. The remainder of the coding provided the basis for calculating interjudge agreement. Disagreements were discussed and resolved, though original scores were retained for calculating the agreement. All the coding was used for the data analysis in other respects, on the grounds that divergences had been discussed and resolved and principles applied retroactively to the coding before the calibration. These procedures resulted in very high interjudge reliabilities on both tests with correlations of .90 or greater in all cases.

In order to determine whether any of the judges tended to score significantly higher or lower than the others, the mean score and standard deviation for each judge were calculated and compared for the totals on both the BASIC and Cognitive tests. In each case ANOVA indicated no significant differences among the judges. Therefore, for those tests that were multiply scored, the judges scores were averaged, while for those tests that were scored by only one judge, that score was used in all analyses.

Results

Fidelity to Metacourse Lessons as Written

Of critical importance in evaluating any educational intervention are questions related to how it is actually implemented. We observed a number of Metacourse, and non-metacourse lessons taught by each teacher in order to assess how the teachers and students responded to the materials. We found that the teachers covered the content of the Metacourse with good fidelity, usually rated around 5, where 7 was described as teaching the material "almost exactly as written" and 4 indicated "the same or very similar content of lesson as written, but adapted and paraphrased". This was consistent with the rating observed in our preliminary study, also about 5. It appears that the lessons were either paraphrased or presented in a manner very similar to how they were originally written.

Teachers were usually rated as covering three quarters of the material in each lesson quite adequately and also as referring appropriately during the Metacourse lessons to the major principles stressed in the Metacourse, ie. the paper computer, purpose, syntax, action, patterns, with the exception of the minimanual which was very rarely mentioned. Thus the minimanual may be an underutilized resource, at least in terms of its integration into the instructors' presentations.

In contrast to the generally good coverage of the lessons and principles of the Metacourse, teachers had much more difficulty in pointing out bridges or applications either to other academic disciplines or problems in real life. Such bridges were noted in only about 10% of the classes, whether they were Metacourse or non-Metacourse lessons.

Homework

The primary purpose for the homework/worksheet analysis was to determine whether or not teachers and students were actually able to use the assignments. The analysis indicated that they were. Students seemed to find the assignments appropriate and of reasonable levels of difficulty.

The secondary purpose was to determine where the Metacourse lessons might be improved. In general, the homework analysis was most useful in detecting problems, not strengths, in the Metacourse and was best interpreted in conjunction with teacher comments. This analysis, along with teacher comments was used in preparing notes to be used for teachers in a future study in which potential problems in implementing the Metacourse could be addressed.

Some specific problems noted during the analysis included the following: (a) a number of students had difficulty hand executing simple programs that included PRINT and LET statements containing expressions; (b) many students did not adopt the pointer that the Metacourse suggested they use to keep track of the current line during hand execution; (c) PRINT/INPUT combinations confused many students -- the PRINT statement preceding the INPUT statement often interfered with hand execution of the INPUT statement; (d) the Assume User Types area of the computer world was not used by some students, perhaps because this was not explicitly modeled by teachers; (d) many students did not attempt to identify patterns in problems, presumably because they did not understand this concept; (e) a common hand execution problem with somewhat more complicated programs was to miss one of the many required steps (e.g., changing a variable in the Variables Area of the Computer World each time it is encountered in a FOR/NEXT loop). Many of these same difficulties manifested themselves later on in the BASIC end of semester test.

Differences between Metacourse and Ordinary Lessons

As in our previous study, observations from the experimental sites were examined for differences in student and teacher behavior when Metacourse lessons were taught as compared to when ordinary lessons were taught. Of particular interest were any major differences in what went on in class during such lessons and whether these seemed disruptive or beneficial. Since in each case the observations of a particular class were made by a single observer who was not blind to the type of class being conducted, these results must be interpreted with caution. However, it should be noted that the observers were generally quite experienced, the variations reported between teachers were not large, and the general pattern of results obtained appears quite reasonable.

There were few surprises in the results we obtained. As in our previous study the teachers seemed somewhat more comfortable when presenting their own lessons (mean rating of 6.9) compared to the new Metacourse material (mean rating of 5.8). However both scores are quite high (7 point scale) indicating general comfort with both old or new material. As before, we found large differences between teachers' styles, with non-Metacourse lessons rated as almost twice as interactive (60% vs. 32%), while somewhat less presentational (26% vs. 15%) than Metacourse lessons. During Metacourse lessons the data suggest that more time was spent in lecture (25% vs. 12%), demonstrations (20% vs. 11%), and in hand execution exercises (21% vs. 9%), and less time in discussion (23% vs. 40%), and "other" exercises (18% vs. 48%). Student behavior during both types of classes was rated as quite responsive, attentive, engaged, and interactive with their teachers (ratings generally over 5 in both types of classes; however, there was apparently more interaction among students during the non-Metacourse lessons (mean rating of 5.4 vs. 3.4). No differences were observed in terms of the nature of such interactions, with both teachers' and students' interactions described as process-oriented and product-oriented about half the time, with very little disciplinary interaction occurring at any time. Students seemed quite prepared for either type of lesson (5.1 for the Metacourse, and 5.7 for non-Metacourse lessons). Finally, despite the differences that were noted, the overall effectiveness of each type of lesson was judged quite high (5.4 for Metacourse, and 6.0 for non-Metacourse lessons).

Group Differences

The findings above seem to indicate that despite differences between normal and metacourse lessons, the Metacourse materials were adequately covered although both the observations and teacher self reports indicate considerable difficulty in making connections between the material and other academic and life problems.

However, assessing the impact of the Metacourse also requires equivalence of the treatment and control groups on other factors that might influence programming achievements, or statistical corrections as needed. Consequently the data were examined by group to check for significant differences extant before the treatment.

Previous Experience with Computers and Computer Programming. Most of the students in both groups had little previous experience with programming languages. Three quarters of the experimental group and 84% of the control group had no previous exposure to BASIC. Those few with experience, however, had minimal exposure (typically a week or two here or there). While 24% of the experimental group had some experience with another programming language compared to only 5% in the control group ($p<.05$), this consisted in all but 2 cases in each group of a brief exposure to LOGO, some years back. Finally a little less than half our subjects in each group (44% in the experimental and 49% in the controls) had a computer in their home. Thus with respect to these types of experiances the groups seemed not substantially different.

Cognitive Pre-Test. The cognitive test was assumed to be an indicator of general ability that might relate to later BASIC performance, and in fact significant positive correlations were obtained in both groups ($r=.69$, $p<.01$) in the expenmental group and ($r=.40$, $p<.01$), in the control group. The Analysis of Variance indicated no significant difference in homogeneity of slopes between the two groups ($F= 1.345$, n.s.).

Overall the control group performed significantly better than the experimental group ($p<.01$), commiting on average about 12 errors compared to 16 errors for the experimental group. The range was from about 6 to 29 errors in the 9 experimental classes compared to 5 to 16 errors in the 13 control classes (A more detailed analysis of subtests is presented later when the issue of transfer is considered.)

Thus the control group appears to be of somewhat higher ability than the experimental group, yet as the data will indicate the experimental group performed significantly better on the BASIC test administered at the end of the course.

Impact of Instruction on Mastery of BASIC

Table1 (next page) indicates the performance of experimental and control groups in terms of mean errors on each of the 16 BASIC Test problems as well as subtotals for four types of problems and the total test. Despite the apparent higher general ability level in the control group the experimental group averaged about 5.5 fewer errors, nearly half a standard deviation overall, than the control group. They produced about 77% correct responses compared to about 66% for the controls (a maximum of 61 errors was possible).

Furthermore, the experimental groups did significantly better than the controls on all major categories of problems, with the smallest advantage coming on the simple one line command problems and the production problems (about a third of a standard deviation), and the largest on the hand execution problems (nearly two thirds of a standard deviation).

Results for individual problems within three of the major categories -- Hand execution, Debugging and Production -- show that the experimental group had significantly fewer errors on every problem except one requiring the use of an IF/THEN statement to create a loop (Problem15). In fact, both groups found it more difficult to write a program using IF/THEN after being asked to create a program that generated identical results but used the more natural FOR loop (Problem 14).

Interestingly, there was also no significant difference between the two groups on the IF/THEN problem within the other major category -- Simple one line commands. The only problems within this category for which there were significant differences included one which asked students to set a variable to a string (Problem 6), one requiring the completion of a FOR/NEXT loop (Problem 8) and one in which students needed to recognize when a GOTO statement was needed (Problem 9). These last two problems (8 and 9) differed from the other problems within this category because partial code was provided that, presumably, needed to be read and understood before being completed. In this respect, perhaps, these problems were similar to the Hand-execution problems.

The BASIC test problems described above often contained the same features or error categories. This allowed us to undertake a finer analysis of how students handled a particular feature across and within various problems. These additional error categories are described below, and the results presented in Table 2.

Table 1. Errors (Mean Number) on Each Problem of the BASIC Test

	Simple One Line Command Problems									
	1	2	3	4	5	6	7	8	9	Total
Exp.	.13	.13	.53	.11	.25	.71	.88	.66	.27	3.67
Cont.	.08	.11	.72	.16	.22	1.14	.88	1.13	.49	4.92
Diff.	.05	.02	-.19	-.05	.03	-.43***	-.00	.47***	-.22***	1.25***
D/S.D										.32

	Hand Execution Problems				DeBugging Problem
	10	11	13	Total	12
Exp.	.28	.74	1.65	2.67	.61
Cont.	.73	1.47	1.94	4.14	1.05
Diff.	-.45***	-.73***	-.29*	-1.47***	-.44***
D/S.D.			.61		.48

	Production Problems				All Problems
	14	15	16	Total	Total
Exp.	1.13	2.83	3.41	7.36	14.31
Cont.	1.60	3.11	5.07	9.77	19.88
Diff.	-.47*	-.28	-1.56*	-2.41*	-5.57***
D/S.D.				.34	.45

Experimental Group (n=120)
Control Group (n=224)

* p<.05
** p<.01
*** p<.001

Simple one line command error categories (Problems 1-9):

1. PRINT (Problems 1-2): The word PRINT is not present or is spelled incorrectly.
2. Quote (Problems 1 and 3): The string following the PRINT command is incorrect or is not preceded by a quotation mark.
3. LET (Problems 4 - 6): The word LET is not present or is spelled incorrectly.
4. Assignment (Problems 4 - 6): The assignment statement following the LET command is incorrect.
5. ACTION (Problems 7 - 8): The expression following THEN in problem 7 is incorrect; the expression between the FOR and NEXT commands in problem 8 is incorrect.
6. GOTO (Problem 9): The GOTO command or the line number following it is incorrect.
7. INPUT (Problem 3): The word INPUT is not present or incorrect.
8. MISC. (Problems 2,3,7,8): Various errors occuring in only one problem.

Hand-execution error categories (Problems 10, 11 and 13):

9. Prompt/Input (Problems 10, 11 and 13): The output generated from INPUT statements does not include the correct prompt and/or input.
10. Output (Problems 10, 11, and 13): The output generated from PRINT statements is incorrect.
11. Minor execution: Any execution errors not covered by other categories.
12. Format: Output is not properly formatted.Debugging error categories (Problems 11, 12, 13):
13. Bug-11 (Problem 11): The correct ("unexpected") output is not generated. This category does not include categories like minor execution, output and so on that were also scored in Problem 11. The only feature considered is the one directly related to debugging.
14. Bug-12 (Problem 12): The correct INPUT statement is not added to the program (as above, other "non-bug" categories are excluded from this category).
15. Bug-13 (Problem 13): The correct ("unexpected") output is not generated (as in category 13, other "non-bug" categories are excluded).

Production error categories (Problems 14, 15 and 16)

16. Counter (Problems 14 and 15): The counter following the FOR command is incorrect.
17. Action (Problems 14 and 15): The major action (PRINT X) is incorrect.
18. FOR (Problems 14 and 16): The FOR statement is missing or incorrect.
19. NEXT (Problems 14 and 16): The NEXT statement (NEXT X) is incorrect or missing.
20. Variables (Problems 14 and 16): Incorrect number or type of variables used.
21. Extraneous (Problems 14, 15 and 16): The program includes unneeded commands that interfere with the running of the program or that make the solution clumsy.
22. Minor syntax (Problems 14, 15 and 16): This category includes errors not covered by other categories that ocur in more than one problem.
23. Misc. (Problems 15, 16): Various error types appearing in only one problem.

Table 2 (next page) presents results from these individual error categories within and across problems. Like the results presented in Table 1, these results indicate that the Metacourse group made significantly fewer errors in each of the four major categories. However, they also indicate that this did not occur uniformly within each major category. That is, the experimental group, although never commiting significantly more errors than the control group, did not perform significantly better than the controls on all error categories.

TABLE 2. Types of Errors (Mean Number) on the BASIC Test

Simple One Line Command Errors (Problems 1-9)

	1 PRINT	2 QUOTE	3 LET	4 ASSIGN	5 ACTION	6 GO TO	7 INPUT	8 Misc.	9 TOTAL
Exp.	.04	.23	.45	.64	.70	.27	.21	1.14	3.67
Cont.	.05	.22	.80	.72	.93	.49	.35	1.36	4.92
Diff.	-.01	.01	-.35***	-.08	-.23**	-.22***	-.14 **	-.22	1.25**

Hand Execution Errors (Problems 10,11,13)

	9 PROMPT-INPUT	10 OUTPUT	11 MINOR EX.	12 FORMAT	13 TOTAL
Exp.	.70	.13	.53	.04	1.40
Cont.	1.87	.08	.60	.06	2.60
Diff.	-1.17***	.05	-.07	-.02	-1.20***

Debugging Errors (Problems 11,12,13)

	13 BUG-11	14 BUG-12	15 BUG-13	TOTAL
Exp.	.48	.26	.78	1.53
Cont.	.76	.48	.78	2.02
Diff.	-.28***	-.221***	.00	-.49***

Production Errors (Problems 14,15,16)

	16 COUNT.	17 ACTION	18 FOR	19 NEXT	20 VAR.	21 EXTRA.	22 MIN.SYN.	23 MISC.	TOTAL
Exp.	.48	.42	.32	.34	.52	1.02	.96	3.30	7.36
Cont.	.58	.57	.64	.76	.74	1.35	1.16	3.98	9.77
Diff.	-.10	-.15	-.32***	-.42***	-.22**	-.33**	-.10	-.68	-2.41**

Experimental Group (n=120)
Control Group (n=224)

* p<.05
** p<.01
*** p<.001

With respect to the "simple one line command errors" (in the first 9 problems) the experimental group did no better than the controls on a number of simple error categories such as PRINT and quote and assigning a value that a variable should take; however, they made about half the errors of the control group when producing simple LET statements (error category 3) and about one quarter less errors than the controls in specifying the action of simple IF/THEN statement or FOR loop (category 5).

As noted above, the experimental groups made significantly fewer hand execution errors on problems that required creating a screen display when given a program (Problems 10, 11 and 13). It appears that the experimental groups' superiority is due to its ability to represent what a prompt and input statement look like on the screen (Error category 9), since there were no differences in the output , minor execution and format error categories (10,11,&12).

If we in turn ignore the previous error types that may occur during hand execution and look specifically at the aspects of execution that are directly related to program bugs, we find that the experimental group is significantly better at detecting unexpected output (Error category 13 from the Garden Path problem). They are no better, however, at detecting the parallelism bug (Category 15). Bug-12 is perhaps a more realistic debugging problem. The students were given a program and simply asked to correct as well as find a bug. In this situation, the experimental group was again significantly better than the non-metacourse group.

The final set of error categories to be considered are those related to the production of whole programs. When given three programming problems, the group with Metacourse training handled FOR/NEXT loops better, was better able to choose the correct number and type of variables to use and produced significantly fewer unnecessary statements (error categories 18 through 21). There were no signficant differences between the two groups on the remaining production error types. As was the case with all other variables analyzed in this study, the control group did not make significantly fewer errors in any of the categories investigated.

Transfer

As noted in our previous reports, while the cognitive posttest provided an occasion to examine possible transfer of cognitive skills from programming instruction it should be recalled that such a finding was not necessarily expected from the Metacourse treeatment. Most research on the impact of programming instruction on cognitive skills has not found substantial gains, except under very special conditions (ie. specific emphasis on transfer throughout a program, one-on-one tutoring, and tests composed of problems formally similar to those encountered in the programming course). In contrast, the Metacourse was designed specifically to advance programming skills rather than to promote general transfer. Nonetheless, the data presented an opportunity to explore the issue within this limited context.

Impact on General Cognitive Skills

As indicated previously the general cognitive tests were composed of 6 separate subtests (See Appendix D). There was almost no variability of the Attribution of Intellectual Responsibility subtest. Both experimental and control groups attributed an average of betwen 7.7 and 7.9

questions (out of a possible 10) to internal factors, both on pre and posttests. Analysis by positive or negative outcomes yielded similar results. Thus most of our sample attributed their successes and their failures primarily to internal causes, both prior to and after the BASIC programming course. This variable was therefore excluded from all further analyses.

Table 3 (next page) presents the performance of each group on each of the other 5 subtests and their total score for both pre and posttests.

As indicated previously the control group made significantly fewer errors on the pretest overall than the experimental group ($p<.001$). Analysis by subtests indicated that this superiority manifested itself on three of the five subtests (conditional reasoning, precise visual description, and repeats and decides), while no initial differences were found on the visual planning or professors/students problems.

Surprisingly the experimental group improved by about three fewer errors on the posttest, compared to virtually no gain for the control group. It is unlikely that this highly significant outcome ($p<.001$) would be due merely to regression to the mean. Analysis of the data from the component tests indicates that this result is due almost entirely to the considerable improvement of the experimental group on the Repeats and Decides subtest, compared to a modest decline in performance on the part of the control group on that same problem. Thus some evidence of transfer was observed, but it occurred only on the problem most closely related in its formal structure to that of producing coherent comands in a programming language.

Teachers' Reactions to the Metacourse

Since this study was conducted within the laboratory sites context described earlier, we received frequent feedback from the teachers concerning the metacourse materials. The teachers were in general quite positive in their evaluation of the revised metacourse. The utility of the visual model of the computer world, the purpose, action, syntax analysis and the emphasis on patterns received the most praise. The length of some lessons, and the difficulty of integrating these long, rather rigid lessons smoothly into their normal course were most often cited as negative features. Many teachers indicated that they wanted more freedom than the tightly controlled conditions of the experiment permitted. It was apparent that infusing new concepts into their normal lessons is a skill not quickly mastered, and while we would expect easier integration "the second time around," future interventions need to address these concerns.

Discussion

The large scale study reported here was undertaken in order to assess the effects of a "metacourse" which was designed to enhance novices' learning of BASIC by providing a small number of key concepts and strategies that, our clinical research revealed, many students fail to acquire on their own (Perkins, Hancock, Hobbs, Martin, & Simmons, 1986; Perkins & Martin, 1986; Perkins, Martin , & Farady, 1986; Perkins, Farady, Hancock, Hobbs, Simmons, Tuck, & Villa, 1986). A pilot study reported in Perkins, Farady, Simmons & Villa (1986), employing an earlier version of the metacourse, although encouraging, produced little hard evidence on its effectiveness. It was hoped that the current study would be able to provide stronger evidence on the issue.

TABLE 3. Errors (Mean Number) on Problems in the Cognitive Pre- and Post-tests.

	Conditional Reasoning			Visual Planning		
	Pretest	Posttest	Gain	Pretest	Posttest	Gain
Exp.	1.71	1.56	.15	6.24	4.92	1.32
Cont.	1.36	1.33	.03	5.26	4.81	0.45
Diff.	.35**	.23	.12	.98	.11	.87

	Professors/Students			Precise Visual Description		
	Pretest	Posttest	Gain	Pretest	Posttest	Gain
Exp.	.72	.76	-.04	2.87	2.63	.24
Cont.	.68	.54	.14	2.02	2.03	-.01
Diff.	.04	.22***	-.18*	.85**	.60*	.25

	Repeats and Decides			Total		
	Pretest	Posttest	Gain	Pretest	Posttest	Gain
Exp.	4.84	3.40	1.44	16.34	13.22	3.12
Cont.	2.95	3.53	-.58	12.34	12.29	0.05
Diff.	1.89***	-0.13	2.02***	4.00***	.93	3.07***

Experimental Group (n=122)
Control Group (n=219)

* p<.05
** p<.01
*** p<.001

As in our previous report it is appropriate to consider the three questions: Did the Metacourse prove "teachable", leading to an implementation that smoothly provided the intended concepts and practice in their use? Did the Metacourse have the hoped-for impact on students' mastery of programming? Did the Metacourse, or the normal programming instruction in the control groups, have a cognitive impact beyond the targeted instruction? Finally questions of possible long term effects, and future directions should also be briefly considered.

Teaching. Once again the data indicate that at least for experienced teachers under conditions of considerable support the Metacourse was quite teachable and could be integrated into the normal curriculum in BASIC. Although the Metacourse lessons produced a variety of differences in teacher style of presentation as well as student behavior, compared to the normal lessons, both teachers and students appeared to adapt well to the new material with a number of indicators pointing to effective classes. Further it may well be the case that some of these differences in observed behaviors (e.g., amount of student-teacher interaction) might be due to the fact that this was the first time the teachers used the materials, and that such differences may diminish as teachers get more familiar with the material.

The evidence on the extent to which the instructors were able this first time through to "infuse" the key Metacourse concepts into their entire course is not as clear. We did find that during the Metacourse lessons teachers referred to earlier key concepts quite often, but the results are not clear from the limited number of non-Metacourse lessons we could observe as to how often these concepts were used in regular classes. We did note a general absence of bridges or applications of these concepts to any areas outside of the programming problems themselves. It seems clear that deliberate efforts have to be orchestrated if such bridges are to be built.

Impact on programming mastery. It is clear that our experimental groups evidenced a considerable advantage in their general competence as beginning BASIC programmers compared to the control groups. Further, the results indicate that this improvement is manifested on a variety of programming skills, such as: use of correct syntax to perform simple operations, ability to trace the actions of a program through hand execution, and the ability to debug and produce simple programs. While the Metacourse did not produce enhanced performance on every problem, e.g., detecting the "parallelism bug" (Pea,1985), it did produce fewer errors on a variety of significant problems typical of those encountered in an introductory BASIC course. What is not so apparent is the cause of the improved performance. A number of possibilities are plausible, and the results may well be due to a combination of factors.

We would hope that the Metacourse with its emphasis on the students' development of mental models through which they understand what the computer does, strategies by which to organize their problem-solving efforts, etc. played a significant role. However, it must be acknowledged that the "laboratory sites" intervention of which this study was a part was an atypical treatment in that the participating teachers received much more support than is normally the case when a new curriculum intervention is introduced. Further, we could not randomly assign teachers to treatments. The laboratory site high school programming teachers were all required to use the Metacourse, while the control group was formed by soliciting volunteer teachers at similar high schools. While the teachers in both groups were experienced teachers of BASIC, it is possible that our treatment teachers were simply a group of exceptionally talented instructors who could produce large gains even with a treatment with which they were unfamiliar. One should note that our study, like most in the field, pits teachers using a treatment program for the first time against controls who employ their "normal" curriculum which they have typically worked through many times.

Another study, currently in progress, should help resolve some of these ambiguities. In this study the revised Metacourse is being used by 9 other teachers in 7 new high schools under conditions that more nearly duplicate normal classroom innovations. That is, these teachers have been given the Metacourse along with some "Metacourse memos" (notes to teachers on experiences teaching with the Metacourse), and are provided with no other supports. The same assessment instrumsnts have been employed as in the previous study. The results from this study should help clarify the significant factors influencing the improved performance in BASIC.

Transfer. Although our results were in general consistent with most of the literature, in that there was little evidence of transfer either on most cognitive tasks or on our affective measure, we did obtain evidence of transfer on one subtest that was structually somewhat similar to a typical programming task. It may also be the case that an affective instrument more specifically focused on students attitudes concerning errors may yet reveal some changes in this realm.

As we have pointed out previously, we believe a kind of tradeoff may exist between teaching for programming competency and teaching for transfer of cognitive skills from programming (Perkins, Schwartz,& Simmons,in press). The metacourse as currently designed still focuses on the development of programming competency rather than transfer of general cognitive skills. We provide little that does not have direct bearing on programming competency. The next section discusses some future directions that may affect this balance.

Future Directions. While the results obtained in this study are encouraging, even the performance of the treatment groups on our BASIC end of semester test left considerable room for improvement (77% correct responses). Further, teachers comments made it clear that, though valuable, the Metacourse in its current 9 lesson format makes both the integration into their normal curriculum, and the infusion of key concepts throughout the term, a considerable challenge. Students still often exhibit an inert, rather than active knowledge of the programming ente[illegible]ise. This "fragile" knowledge does not permit optimal use in contexts when it should be app[illegible] [illegible]ven within the discipline of programming let alone in other problem solving realms. The pr[illegible]amming research group at ETC is currently developing a "second generation metacourse", to address some of these concerns.

This second generation metacourse is built upon a "language" rather than a "lesson" model. Thus rather than a collection of fully worked out lessons, a number of short modules introduce the key concepts, while other modules focus explicitly on how such concepts can be practiced or "infused" throughout the semester. Finally a number of optional modules attempt to illustrate how bridges can be made to academic and "real life" problems, thus encouraging transfer to more general cognitive skills. While providing teachers more freedom to utilize the Metacourse in their own fashion, it is designed to make more explicit and hopefully available to both teachers and students the core elements of our approach and the rationale behind them. Such aids as an "animated computer world" with an agent performing simple operations on information, icons, metaphors, and posters are all being considered as metacognitive supports. This second generation language model metacourse will be empirically tested in the near future.

What has become more apparent as we have interacted with teachers is that many of them share a goal of teaching their students in some sense "what computers and programming languages are really all about", "how they can be used as problem-solving tools" and not just the syntax of a particular programming language like BASIC. It may be the case that the achievement of this goal through the design of a second generation metacourse is also more compatible with the development of general cognitive skills, and that a powerful pedagogy of programming focusing on attention to mental models, strategies, etc. may lead to more success at achieving both programming competence and transfer.

References

Beveridge, M. & Parkins, E. (1987). Visual representation in analogical problem solving. Memory and Cognition, 15, 230-237.

Bonar, J. & Soloway, E. (1985). Pre-Programming knowledge: A major source of misconceptions in novice programmers. Pittsburgh, PA: Learning Reserach and Development Center. (ERIC Document Reproduction Service No. ED 258 805.)

Clement, J., Lochhead, J., & Monk, G. (1981). Translation difficulties in learning mathematics. American Mathematical Monthly, 88, 26-40.

Clements, D.H. (1985, April). Effects of Logo programming on cognition, metacognitive skills, and achievement. Presentation at the American Educational Research Association conference, Chicago, Illinois.

Clements, D.H., & Gullo, D.F. (1984). Effects of computer programming on young children's cognition. Journal of Educational Pyschology, 76(6), 1051-1058.

DuBoulay, B. (1986). Some difficulties of learning to program. Journal of Educational Computing Research, 2(1), 57-73.

DuBoulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices. International Journal of Man-Machine Studies, 14, 237-249.

Dweck, C.S., & Licht, B.G. (1980). Learned helplessness and intellectual achievement. In J. Garbar & M. Seligman (Eds.), Human Helplessness. New York: Academic Press.

Ehrlich, K., Soloway, E., & Abbot, V. (1982). Transfer effects from programming to algebra word problems: A preliminary study (Report no. 257). New Haven: Yale University Department of Computer Science.

Gentner, D., & Stevens, A.L. (Eds.). (1983). Mental Models. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Johnson-Laird, P.N. (1983). Mental Models. Cambridge, Massachusetts: Harvard University Press.

Joni, S.A., & Soloway, E. (1986). But my program runs!: Discourse rules for novice programmers. Journal of Educational Computing Research, 2(1), 95-125.

Kurland, D.M., Clement, C., Mawby, R., & Pea, R.D. (1987). Mapping the cognitive demands of learning to program. In D. N. Perkins, J. Lochhead, & J. Bishop (Eds.), Thinking: The second international conference (pp. 333-358). Hillsdale, New Jersey: Erlbaum.

Kurland, D.M., Pea, R.D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. New York: Bank Stree[illegible] College of Education, Center for Children and Technology. Also, Journal of Educational Computing Research, [illegible]ess.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. Educational Researcher, 14, 14-29.

Mawby, R. (1987). Proficiency conditions for the development of thinking skills through programming. In D. N. Perkins, J. Lochhead, & J. Bishop (Eds.), Thinking: The second international conference (pp. 359-371). Hillsdale, New Jersey: Erlbaum.

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

Mayer, R.E. (1981). The psychology of how novices learn computer programming. Computing Surveys, 13(11), 121-141.

Mayer, R.E. (1985). Learning in complex domains: A cognitive analysis of computer programming. The Psychology of Learning and Motivation, 19, 89-130.

Pea, R.D. (1986). Language-independent conceptual "bugs" in novice programming. Journal of Educational Computing Research, 2(1), 25-36.

Pea, R. D., & Kurland, D.M. (1984a). On the cognitive effects of learning computer programming. New Ideas in Psychology, 2(2), 137-168.

Pea, R.D., & Kurland, D.M. (1984b). Logo programming and the development of planning skills (Report no. 16). New York: Bank Street College.

Perkins, D.N. (1986a). Knowledge as Design. Hillsdale: new Jersey: Lawrence Erlbaum Associates.

Perkins, D.N. (1986b). Knowledge as design: Teaching thinking through content. In J. B. Baron & R. S. Sternberg (Eds.), Teaching thinking skills: Theory and practice (pp. 62-85). New York: W. H. Freeman.

Perkins, D.N., Farady, M., Hancock, C., Hobbs, R., Simmons, R., Tuck, T., & Villa, E. (1986). Nontrivial pursuit: The hidden complexity of elementary Logo programming (Tech. Report no. 86-7). Cambridge, Massachusetts: Harvard Graduate School of Education, Educational Technology Center.

Perkins, D.N., & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In E. Soloway & S. Iyengar (Eds.), Empirical studies of programmers (pp. 213-229). Norwood, New Jersey: Ablex.

Perkins, D.N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. Journal of Educational Computing Research, 2(1), 37-56.

Perkins, D.N., Martin F., & Faraday, M. (1986). Loci of difficulty in learning to program (Tech. Report no. 86-6). Cambridge, Massachusetts: Harvard Graduate School of Education, Educational Technology Center.

Perkins, D.N., Schwartz, S., & Simmons, R. (in press). Instructional strategies for the problems of novice programmers. In Mayer, R. (Ed.), Teaching and learning computer programming: Multiple research perspectives. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Salomon, G., & Perkins, D.N. (in press). Transfer of cognitive skills from programming: When and how? Journal of Educational Computing Pesearch.

Sleeman, D., Putnam, R., Baxter, J., & Kuspa, L. (1986). Pascal and high school students: A study of errors. Journal of Educational Computing Research, 2(1), 5-23.

Soloway, E. & Ehrlich, K. (1984). Empirical studies of programming knowledge. IEEE Transactions on Software Engineering, SE-10(5), 595-609.

Soloway, E., Lochhead, J., & Clement, J. (1982). Does computer programming enhance problem solving ability? Some positive evidence on algebra word problems. In R. Seidel, R. Anderson, & B. Hunter (Eds.), Computer literacy. New York: Academic Press.

Zelman, S. (1985, April). Individual differences and the computer learning environment: Motivational constraints to learning LOGO. Presented at the American Educational Research Association Annual Meeting, Chicago, Illinois.

Appendix A

Paper Computer World

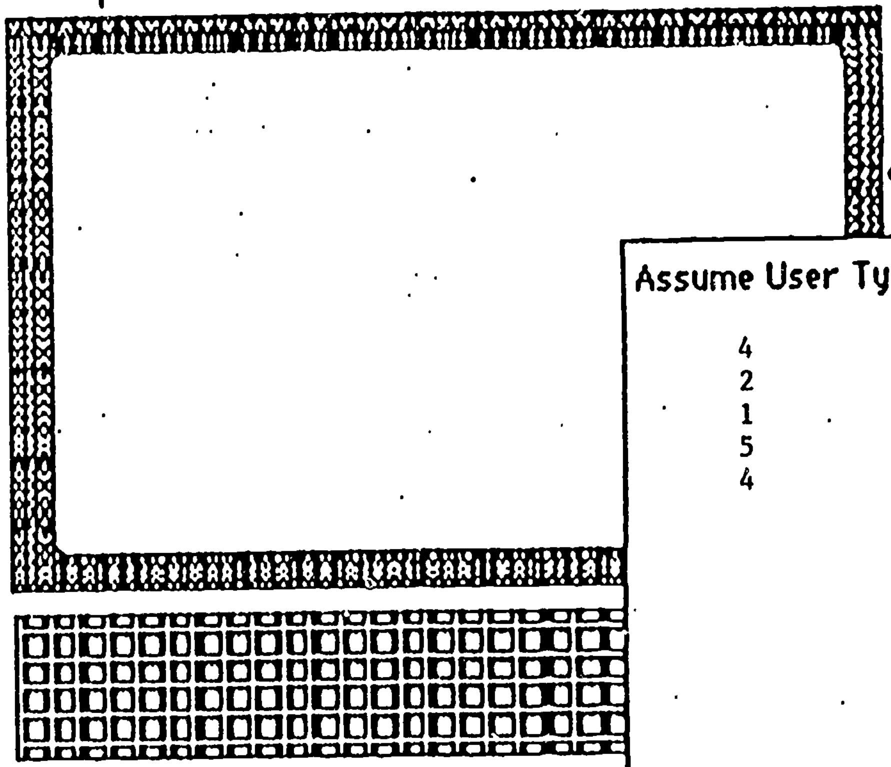
the program. (c) Change the f[illegible] print statement so that it explains the output.

The Computer World

Program Area

```
10   PRINT "HOW MANY PEOPLE?"
20   INPUT P
30   G = 0
40   PRINT "PLEASE ENTER THE NUMBER"
50   PRINT "OF SLICES OF PIZZA THAT
60   PRINT "EACH PERSON WANTS"
70   FOR I = 1 to P
80   INPUT N
90   If N > 3 THEN G = G ÷ 1
100  NEXT I
110  PRINT "THE ANSWER IS";G
120
130
140
```

Variables Area



Assume User Types:

4
2
1
5
4

Appendix B

Example of Purpose, Action, and Syntax
from Mini-Manual

III. LET

PURPOSE: You use LET in a program to make the computer save information in its memory for later use. The information is stored in a variable. Think of a variable as a box with name. Boxes with names ending in $ hold only strings. Boxes without the $ hold only numbers.

SYNTAX	ACTION: SCREEN DISPLAY	ACTION: WORK SPACE
LET Variable Name = Number		
for example: LET X = 5		5 is stored in the X box
Is it really there? PRINT X	5	
LET Variable Name = Legal Numerical Expression		
for example: LET Z = 5 * (1 + X)		1 + 5 is 6 5 * 6 gets you to 30 30 is stored in the Z box
Is it really there? PRINT Z	30	
LET Z = Z - 2		30 - 2 = 28 28 is the new value for Z. which is stored in the Z box
Is it really there? PRINT Z	28	

SYNTAX	SCREEN DISPLAY	WORK SPACE
LET String Variable Name = String		
for example: LET A$ = "SIZE"		SIZE is stored in the A$ box.
LET B$ = "/"		/ is stored in the B$ box.
Are they really there? PRINT A$	SIZE	
PRINT B$	/	

Note: You can leave off the LET statement, and get the same result.

for example:

X = 2	2 is stored in the X box.
D$ = "DAYS"	DAYS is stored in the D$ box

Appendix C

Example of a "Pattern" from the Mini-Manual

PATTERNS

Summing Pattern

Purpose: To find the sum of a series of numbers

Structure:

```
...
LET T = 0
...
<begin loop>
    ...
    LET T = T + <number to add>
    ...
<end loop>
```

Example:

```
10 REM ADDS 5 NUMBERS
20 LET T = 0
30 FOR I = 1 TO 5
40   PRINT "ENTER A NUMBER"
50   INPUT N
60   LET T = T + N
70 NEXT I
```

Action: The summing variable (in this case, T) is first set to 0. Each time the loop body is executed, another number is added to the summing variable. When the loop is exited, the summing variable contains the total..

Counting Pattern

Purpose: To count the number of times something is true

Structure:

```
...
LET C = 0
...
<begin loop>
    ...
    IF <condition> THEN LET C = C + 1
    ...
<end loop>
```

Example:

```
10 REM COUNTS A'S
20 LET C = 0
30 FOR I = 1 TO 5
40   PRINT "ENTER A GRADE"
50   INPUT G$
60   IF G$ = "A" THEN LET C = C + 1
70 NEXT I
```

Action: The counting variable (in this case, C) is first set to 0. Each time the loop body is executed, you test a condition. If that condition is true, the counting variable is increased by 1.

Appendix D

Cognitive Skills Pre-/Post-Test

Version A

Name: ______________________________ Date: _________

GENERAL INSTRUCTIONS

Our class has been chosen to contribute to some very important educational research. A group of people at Harvard University is attempting to discover the best techniques of teaching BASIC programming so that it will be more interesting and understandable to students.

This post-test, given now at the end of the course, will help them to determine the differences among all of you who will be participating.

The Harvard Group wants you to know how much they appreciate your help and they are looking forward to your comments and the wonderful information they will gather through your efforts.

This is a 40-minute test comprised of a short questionnaire and a number of problems.

Do not start the test until your teacher says "go".

You may work through the test as quickly as you want. After finishing one part, go on to the next.

However, we want to be sure you try all parts. So after the time for a particular section is up, the teacher will say, "please go ahead to section A, B, C, D, E, or F (whatever it is) now if you haven't already".

You can go back to work on a previous section if you have extra time.

Ask any questions you have now.

A. CHOOSE ONE ANSWER 5 Minutes

Pick the answer that best describes what happens to you or how you feel. There are no right or wrong answers.

1. When you read a story and can't remember much of it, is it usually
 ____a. because the story wasn't well written, or
 ____b. because you weren't interested in the story?

2. If a teacher says to you, "Your work is fine", is it
 ____a. something teachers usually say to encourage pupils, or
 ____b. because you did a good job?

3. Suppose you weren't sure about the answer to a question your teacher asked you, but your answer turned out to be right. Is it likely to happen
 ____a. because she wasn't as particular as usual, or
 ____b. because you gave the best answer you could think of?

4. When you read a story and remember most of it, is it usually
 ____a. because you were interested in the story, or
 ____b. because the story was well written?

5. If the teacher didn't pass you to the next grade, would it probably be
 ____a. because she "had it in for you", or
 ____b. because your school work wasn't good enough?

6. Suppose you don't do as well as usual in a subject at school. Would this probably happen
 ____a. because you aren't as careful as usual, or
 ____b. because somebody bothered you and kept you from working?

7. If a boy or a girl tells you that you are bright, is it usually
 ____a. because you thought up a good idea, or
 ____b. because they like you?

8. Suppose you became a famous teacher, scientist or doctor. Do you think this would happen
 ____a. because other people helped you when you needed it, or
 ____b. because you worked hard.

9. Suppose you are showing a friend how to play a game and he has trouble with it. Would that happen
 ____a. because he wasn't able to understand how to play, or
 ____b. because you couldn't explain it well?

10. If you can't work a puzzle, is it more likely to happen
 ____a. because you are not especially good at working puzzles, or
 ____b. because the instructions weren't written clearly enough?

6 Minutes

B. DAYS OF THE WEEK

1. If the day before the day after tomorrow is Sunday, what is the day before yesterday? ______________.

2. Suppose a week had no Wednesday. If today is Tuesday, the day after tomorrow is what? ______________.

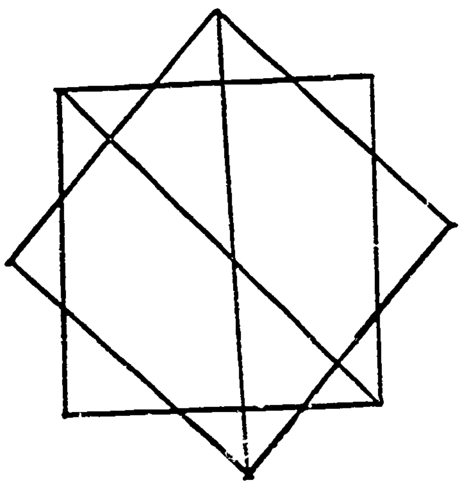
3. Suppose today is the day before Tuesday. What is the day before yesterday? ______________.

4. If tomorrow were Wednesday instead of Sunday, yesterday would be Monday instead of what? ______________.

5 Minutes

C. VISUAL PUZZLE

How many triangles does the figure below contain? ____________.



4 Minutes

D. FORMULA QUESTION

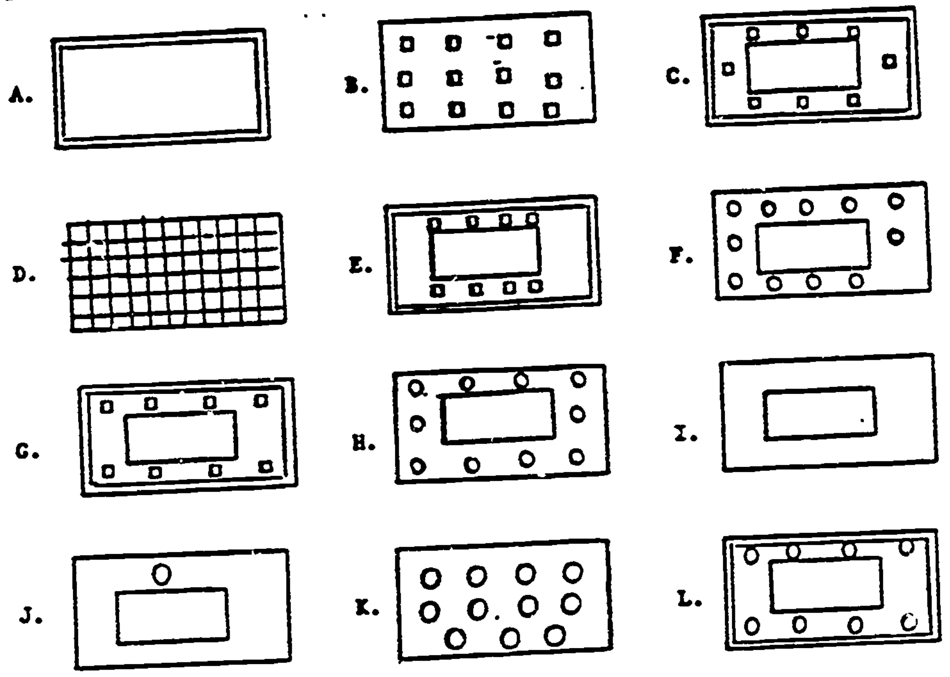
For every 3 people who drink coffee, 1 person drinks tea. Suppose C stands for the number of people who drink coffee. T stands for the number of people who drink tea. Circle the equation that states the relationship between how many people drink coffee and how many people drink tea.

1 = C/T T = 3C C = T/3 C = 3T 3 = C - T

8 Minutes

E. DESCRIPTION

Examine carefully the pictures of the items shown in the following figure. Write a description of Item "G" to someone, so that he/she could pick out "G" from all the other shapes. The other person's paper doesn't have the letters beside to identify each figure, and the figures are all mixed up. Write out your descriptions of "G" below.



F. REPEATS AND DECIDES

10 Minutes

For the next problem, you have to learn about repeats and decides before you do the problems.

REPEAT: A repeat is an instruction to do something over and over again, until some condition is met. For instance:

> REPEAT - Jump up and down until you've done it 17 times.

DECIDE: A decide is an instruction to choose which one, of two or more conditions exist, and then doing something once, based on this choice. For instance:

> DECIDE - Decide if you have homework. If you do, stay home and do it. If you don't, then go see a friend.

WRITING DIRECTIONS USING REPEATS AND DECIDES: You can write directions using repeats and decides. Here is on example that uses one repeat and one decide instruction.

Suppose you want to add up all the money in everybody's pocket in the room and shout "Hurrah!" if the total is over $100.
Here are the directions:

> REPEAT - Add on the money in the next person's pocket until you have covered all the people in the room.
>
> DECIDE - Decide if you have more than $100. If you do, shout, "Hurrah!". If you don't, then be quiet.

Other problems may require more than one repeat and/or decide.

Now go to the next page for the problem.

It's Saturday and you want to go out and see the movie called: "The Crazy Computer". You have a newspaper that lists the movie theaters alphabetically with what is playing at each. If you can't find the movie anywhere then turn on the T.V. to a ball game. If you find the movie then start calling your friends until one says he or she will go with you. If you find one willing to go, then make a date and go. If not stay home and watch the ball game.

If you have extra time, go back and complete sections you did not finish or check your work.

Appendix E

BASIC Test

NAME ________________________________ DATE _______

DIRECTIONS — Work quickly but carefully on the following problems. If you get stuck on one problem, go on to the next.

Write the Basic commands that will cause the following to happen. An example is done for you below.

EXAMPLE. The following word appears on the screen:

Hello

1. The following message is to be printed on the screen:

The answer is cat

2. The number 5 is to appear on the screen.

3. The program asks the user for a number. (The number will be stored in a variable.)

4. The variable X is set to the value 2.

5. The variable X is set to 3 more than Y.

6. A variable (give it whatever name you like) is set to the following:

THE ANSWER IS KNIGHT

7. When the value of Z is greater than 3, the following message is printed: L = 5

Complete the next 2 programs by filling in the blanks with the appropriate code.

8. This program uses a FOR statement to print the number 6 ten times. (Fill in the blanks.)

```
10 FOR X = 1 TO ___
20 PRINT ___
30 ______________
```

9. The following is part of a larger program. What is needed in this part of the program so that lines 220-230 are not executed? (Fill in the blank.)

```
210 ______________
220 Print "No"
230 Print "No"
240 Print "Yes"
```

10. During a run of the following program what will appear on the screen? Suppose the user enters 16 for A.

```
10  PRINT "HOW OLD ARE YOU?"
20  INPUT A
30  IF A=15 THEN PRINT "TOO OLD"
40  IF A>15 THEN GOTO 60
50  IF A<15 THEN PRINT "LITTLE KID"
60  PRINT "RETIRE FROM SCHOOL"
70  END
```

screen display

11. During a run of the above program what will appear on the screen? Suppose the user enters 14 for A.

screen display

12. This program asks the user to enter the number of slices of bread he/she has eaten and then shows on the screen the total number of calories the bread contains. In this program S stands for number of slices of bread, and C stands for calories. (Assume there are 70 calories in a slice of bread.)

The program doesn't work. The output is not correct; it is always:

```
Number of slices
Total calories in bread
0
```

What is wrong with the program? Find the error(s) and correct the program.

```
100 PRINT "Number of slices"

110 LET C = S * 70

120 PRINT "Total Calories in bread"

130 PRINT C

140 END
```

13. The following program was written to calculate weekly pay by multiplying hours worked times hourly wage. P stands for pay, H for hours, and W for wage. The user of this program types in 10 for hours and 4 for wage. During the run what will the screen display show?

```
10  LET P = H *  W

20  INPUT "HOURS "; H

30  INPUT "WAGE "; W

40  PRINT "PAY = "; P

50  END
```

screen display

You will be writing 2 programs that do the same thing; they print out on the screen the whole numbers from 1 to 10. The output will look like this:

```
1
2
3
4
5
6
7
8
9
10
```

In your first program, use a FOR/NEXT loop. In your second program create the loop using an IF statement.

14. PROGRAM WITH FOR/NEXT LOOP	15. PROGRAM WITH IF STATEMENT
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________

16. Each day, a grade school class of 20 students counts the number of cartons of milk needed by the class. Write a program that will: 1) for each of the 20 students ask how many cartons of milk he or she will drink, and 2) calculate the total number of cartons needed by the class and print the total on the screen.

Appendix F

Classroom Observation Sheet

OBSERVATION INSTRUMENT

Observer: ________________ Date: ___________
School: __________________________
Teacher: _________________________
Beginning time: ______ ending time: __________
total class time: __________

control: ____
experimental: ______ metacourse lesson # ____
ordinary class _____
taped? yes ____ no ____

1. Write out an outline of the class as things happen. Try to note the essential features such as major points made, method of presentation, practice time, etc. Revise after class if necessary.

Instruction: Pease describe the behaviours indicated below on a 7 point scale. For most examples, there are definitions and reference points given. Feel free to assign ratings between the reference points, but use whole numbers only (no fractions or decimal points). If a question is unanswerable or somehow not applicable, please mark an "x" in the space. (Where fitting, a brief explanation would be appreciated).

2. ____ Amount of interaction between teacher and students -

7	- lots of interaction between students and teacher
4	- average amount of interaction
1	- little interaction

3. ____ Amount of broadly beneficial interaction among students -

7	- much
4	- some; a few comments between students
1	- none

4. ____ Amount of deleterious interaction among students

7	- much
4	- some; the usual amount of fooling around
1	- none

5. ____ Student responsiveness -

7	- students respond readily
4	- students need to be coaxed
1	- students are unwilling to participate, to answer or to ask questions, etc.

6. ____ Attentiveness of students -

7	- highly attentive to material/instruction
4	- somewhat attentive, but also talking, etc
1	- inattentive — talking, doing other work

7. ____ Percentage of students who seem to be engaged in class activities

8. Teacher - student interaction (Give percentages)

	Teacher's role	Students' role
a. Process oriented	_______	_______
b. Product oriented	_______	_______
c. Disciplinary	_______	_______

9. ____ Preparedness of students -

7 - students seem very well prepared and ready to move on
4 - students seem adequately prepared
1 - students seem to lack understanding requisite to continuing

10. ____ Write in the approximate percentages of the total time spent in:

____ lecture
____ questions and discussion
____ demonstration
____ hand execution exercise
____ other exercises

____ (total should be 100%)

lecture - teacher speaks
demonstration - teacher demonstrates some aspect of programming either on board or with overhead
hand-execution - students/teacher go over program line by line
other exercises- ____
____
____

11. ____ Teacher-material interface -
7 - teacher seems very comfortable and conversant with material
4 - teacher seems adequately comfortable with material
1 - teacher seems uncomfortable with material

12. ____ Mode of instruction
Give percentages.
____ - highly interactive; teacher often asks for comments, talks to and with students
____ - combination of interactive and presentational
____ - presentational; teacher describes, demonstrates, models

13. _____ Subjective, holistic rating of total effectiveness of instruction

13a. Comment - ____
____
____

14. ____ Reference to other lessons -

7	- much reference to earlier lesson(s)
4	- some mention of earlier lessons(s)
1	- no mention of earlier lesson(s)

14a. If ideas from earlier lesson(s) were referred to, note which
____
____

15. Check: (use 1 for "yes" or 2 for "no") Were there any bridges or applications stressed for which Basic Programming principles are used in:

a. _____ other academic disciplines; which one(s): ______________
____
____

b. ____ problems in real life, outside school; which one(s): ____
____
____

How? Please note: ____
____
____
____

For experimental lesson classes only

Score using the following scale:
7: guided use, using it a lot
4: appropriate, right on target
1: no mention

16. Which basic principles from the Metacourse were referred to:

a. _____ the paper computer
b. _____ purpose, syntax, action
c. _____ interaction between the user and the computer
d. _____ the minimanual
e. _____ patterns
f. _____ program production using metacourse principles
g. _____ other: ______________________________

17. _______ Percentage of the important material in this lesson that was adequately covered.

Fidelity of lesson observed to lesson as written -
7: taught almost exactly as written
4: the same or very similar content of lesson as written, but adapted and paraphrased
1: almost unrecognizable as the same lesson

18. ____ Introduction
19. ____ Lecture
20. ____ Participation
21. ____ Explanation
22. ____ Exercise

Appendix G

Student Questionaire

Previous Computing Experience NAME ____________________

Please answer the following questions concerning your experience with computers. If you have a hard time answering any of these questions just give it your best guess. The information will help us in our research, and we appreciate your help with this project.

1) Have you ever taken a BASIC programming course before this semester?

Yes ____ No ____

2) If you answered yes to number 1: When did you take the course(es), how long did the course(s) last and how often did the class(es) meet? (For example, you might have had a BASIC course at a camp this past summer that lasted for 6 weeks and met 3 times a week for 2 hours each class.)

3) Have you ever taken another programming course in a language other than BASIC?

Yes ____ No ____

4) If you answered yes to question 2: What language(s) have you studied?

Logo ____ If yes, how much?____________

Pascal ____ If yes, how much?____________

Other __________ (name of language) If yes, how much? _________

5) In what grade were you first introduced to computer languages (Logo, Basic, etc)? ___________

6) In what grade were you first introduced to computer applications(word-processing, data bases, spreadsheets)?___________

7) Is there a computer in your home? If so, 1) what kind is it, 2) how often do you use it, and 3) what 2 things do you use it for the most (for example, games, programming, wordprocessing)?