PROGRAMMING FOR LEARNING IN MATHEMATICS AND SCIENCE

Sylvia A. S. Shafto, Associate
Learning and Instruction Division
U.S. Department of Education
1200 19th St. NW,Washington, D.C. 20208

Abstract: This paper presents a learning-research based argument for the integration of computer programming into the science and mathematics curricula in pre-college education as well as college. Students who generate solutions to science and mathematics problems develop a procedural understanding of the fundamental theories of these disciplines. Students should be taught to use programming languages for these solutions for the same reasons they are taught the universal tools of arithmetic and algebra, and because only a computer provides the means to describe solutions in explicit, correct, and executable form. Programming should be integrated into all mathematics and science teaching from the earliest years. In precollege education, programming should be taught over a period of eight to ten years, rather than as a 6-12 week separate topic, and should be matched to the level of complexity of the science and mathematics content.

Individual efforts are made in college courses to teach concepts of physics, engineering, statistics and calculus to students by having them write programs to solve problems. Although some colleges are requiring math majors to take introductory computer science, engineering schools are presently the primary institutions where consistent effort is made to use programming as an integral part of teaching the science.

The present concern about a crisis in math and science education centers around evidence that even the best students entering college are unable to apply fundamental concepts of mathematics or science to solve relatively simple problems (Brumby,1979; Champagne et al., 1982; Larkin, 1983; Nickerson et al., 1984).

This paper will discuss ways in which children can learn science and mathematics better by writing computer programs. Instructional research strongly suggests that writing a program to represent a theorem in mathematics or a process in science can improve the programmer's ability to generalize and apply the new theorem or process. For example, consider the physical law: Force = Mass x Acceleration. A high school physics student who translates this statement into a BASIC or PASCAL procedure, including input variables for Mass and Acceleration, will have analyzed this formula into its component variables and will have rehearsed and attended to the relations among these variables and their units of measurement. The student will have constructed a practical computational tool for solving an indefinitely large class of homework problems. The student may also have created a prototype which can be used in writing other programs to solve other problems, for example, using Distance = Rate x Time, Voltage = Current x Resistance, or Temperature = Pressure x Volume. The student may even be progressing toward a grasp of the useful but abstract concept of a multiplicative law, and developing an appreciation for the constraints among the variables in this type of law.

In this paper I want to answer the following questions:

1. What is the relation of computers and programming languages to problem solving?
2. How can computers be used in teaching problem solving in mathematics and science?

ACM-0-89791-178-4/86/0002/0296 $00.00

3. How can we more effectively teach computer programming as a problem-solving tool in the pre-college curriculum?
4. Is it feasible to incorporate computer programming into pre-college science and mathematics classes?

COMPUTERS, PROGRAMMING LANGUAGES, AND PROBLEM SOLVING

Computers and programming languages were originally invented to help people solve difficult problems. A general-purpose programming language can be used to formulate the solution to any solvable problem, and computers have been applied with dramatic success in many problem-solving contexts. There is no evidence, however, that computer programming has magical powers. There is no convincing evidence, for example, that writing programs will automatically improve a student's creativity or general reasoning ability or higher-order cognitive skills. The following studies all suggest that we should be skeptical of such claims.

Glaser (1984) summarizes a wide range of attempts to teach general thinking or problem solving skills. His conclusions are generally skeptical not just of computer-based methods, but of almost all methods that purport to improve general reasoning skills. He questions the theoretical soundness of attempts to teach general heuristics for reasoning across a wide variety of content domains. Current theory and findings in cognitive science, developmental psychology, and the study of human intelligence emphasize the interaction between the development of problem-solving and learning skills, and the acquisition of specific knowledge.

Glaser observes, "The current literature poses a dilemma between instructional emphasis on general domain-independent skills or domain-specific skills.... A central issue for theory and experiment in resolving this issue will focus on the transferability of acquired knowledge and skill."

If broad, teachable, domain-independent thinking skills exist, we should adopt the tactics of the general methods programs. If humans show limited capability to transfer and generalize thinking skills, then training in the context of specific knowledge domains is called for. A third possibility is that effective problem-solving and learning strategies can be taught explicitly along with specific knowledge. The research evidence available today, however, fails to establish the existence of general reasoning and problem-solving skills that can be explicitly taught.

The Bank Street College of Education (Pea, 1984; Pea et al., 1984; Pea & Kurland, in press) carried out a series of studies over several years which specifically addressed the question, "Does learning to program affect other cognitive skills?" Their general conclusions agreed with Glaser's. They found no evidence for the claim (often associated with Seymour Papert) that "teaching the computer" causes increased insight about one's own thinking and better thinking in general. Papert's claim that gains in general reasoning ability come from exposing children to mind-sized bites of powerful ideas, such as modularity, variables, and recursion, was not supported. Evidence was also lacking for general improvements in planning ability across domains, or debugging as a generic problem solving skill.

There are many limitations to the Bank Street studies, but one clear point is that there is no reason to believe that dramatic improvements in general reasoning or problem solving skills will automatically result from learning to program using LOGO or any other language.

As Roy Pea (1984) notes, "From a functional perspective we may see that powerful ideas are no more attributes inherent 'in' LOGO than powerful ideas are inherent 'in' written language. Each may be put to a broad range of purposes. What one does with LOGO ... or any symbol system is an open matter."

The Bank Street researchers and Stein & Linn (in press), however, have argued that computer programming may yet prove useful for teaching general planning skills. To determine whether or not programming is effective in teaching such general skills, it is important to use instructional methods that emphasize "strategic planning, testing, and revision" (Stein & Linn, in press). Most computer-based courses may not be long enough or demanding enough to produce the intended benefits. The ACCCEL program (Dalby et al., 1984) is a set of techniques and guidelines intended to "encourage cognitive outcomes." For example, students make an action diagram of their program before beginning to write code, and they must generate at least two hypotheses to explain a bug before trying to fix it.

Dalby et al. (1984) argue that there are many features of computer programming that make it a good vehicle for learning how to solve complex problems. Microcomputer-based systems are interactive and provide fast

feedback. CAI programs provide reproducible, consistent results. Programming itself is a complex problem-solving task in which there are multiple correct solutions. The computer provides a challenging, yet controllable, environment in which students can practice cyclical procedural skills, such as planning, testing, and debugging. Students can engage in autonomous learning, in an environment which accommodates a wide range of individual differences.

Thus, while acknowledging the research which shows severe difficulties with transfer and generalization, these researchers, like Glaser (1984), hold out hope that somehow generalized reasoning or learning strategies can be taught along with specific knowledge -- if only the instructional method has "The Right Stuff."

CURRENT USES OF COMPUTERS TO TEACH PROBLEM SOLVING

Even if learning to use a program does not lead directly to improved reasoning and learning skills, there are other ways computers can be used to enhance mathematics and science education. Many of these uses depend on "canned" programs, including CAI programs (Orlansky, 1983), microworlds, and simulations. Some are written by teachers, using authoring languages like LOGO (Feurzeig & White, 1984) or PILOT, while others are commercial products. Recent examples can be found in Chemistry (Zeisler, 1985; Sparrow, 1985), Biology (Lu, 1985; American Biology Teacher, Jan., 1985), and Mathematics (Carrier et al., 1985). These approaches share the property that they do not require the students to write their own programs to solve substantive problems.

As reflected in these publications, the dominant use of microcomputers in science and mathematics education is to present quick summaries of data in the classroom and laboratory. Data from real or simulated experiments in Physics, Chemistry, or Biology are analyzed and displayed. The computer is used to calculate the mean of a set of numbers, project the future size of a simulated population, or produce a graph of temperature changes over a period of time. Pre-written programs help simplify the teacher's task and reduce the drudgery of calculation for the student. Data can be accessed faster and displayed in dynamic and varied formats that are not possible without computers. There are good arguments as to why these types of applications should help in teaching, and their educational value is being investigated (e.g., Wiser et al., 1985; Smith et al., 1985).

It is not yet clear, however, how these programs -- designed to interact with students -- will be best used in teaching science and mathematics. It is especially unclear whether a student who only changes the initial values of the number of arctic hare and lynx in a simulation program learns anything about the underlying model of population dynamics by observing the output of the program. What I would like to see in education is students writing their own programs to carry out simulations, after having studied and discussed the underlying algebraic model of population interaction.

COMPUTER PROGRAMMING AS A PROBLEM-SOLVING TOOL

According to Spain (1982): "The process of developing a simulation model forces the investigator to describe the system in simple terms. When working with the model in this way, the investigator must take into account details about the system which might otherwise go unnoticed. The objective of any kind of model is to provide a means for obtaining new insights into the operation of a system. Simulation assists in this regard by permitting experimental interaction with the model to produce verifiable responses.... The computer is a powerful laboratory tool ideally suited for developing and testing biological concepts.... Real understanding of a system often comes more from the process of model development than from examination of the simulation data that the model produces."

This quotation captures the main point I want to make: Computer programming consists of understanding a problem, creating a solution, and representing that solution using a programming language. The educational value of programming often comes more from the process of writing the program than from the results of running it.

Although the literature on educational software might imply that this is not an important educational use of computers, it is not entirely unknown in science and mathematics classes in middle school and high school. Many teachers have encouraged students to write their own programs for representing important theorems and processes. However, this has been done almost exclusively on an individual teacher basis. Although the professional community has called for the integration of computer programming into the

mathematics curriculum (Kilpatrick & Wilson, 1984), no concerted effort has been made to achieve this goal.

I believe this integration has such a real educational value that a kindergarten through high school plan should be developed, so that students at any point in their education will be able to write programs of comparable sophistication to the science and mathematics they are learning. Starting with the simplest arithmetic applications in kindergarten, students could write programs which emphasize the significance of an operator such as the plus sign in creating a counting series. By middle school, students should be sophisticated enough in programming to be able to write programs representing physical laws they are being taught. After nine or ten years of using a programming language, most students should be successfully writing programs to implement the theorems and processes presented in high school mathematics and science courses.

As an example of one major effort to integrate Biology course material and computer programming, Spain (1982) has written a textbook designed to encourage programming simulations for biological phenomena. This material is appropriate for students in high school and college Biology courses. Rather than providing a collection of programs a student can copy, Spain emphasizes describing and explaining important models in Biology. He lays the substantive groundwork for the student, who is given the responsibility of writing her own programs. (Spain assumes a suitable level of familiarity with BASIC.) Pedagogically, the student is challenged to decompose each model into its elements and restate it as a working computer program.

Du Boulay (1980) used computer programming to aid student teachers in the learning of mathematics facts and concepts, and to give them practice in mathematical problem solving. Student teachers, like students, sometimes have difficulty with routine calculations involving fractions, decimals, squares, and square roots. They also have a variety of problems that do not show up on conventional tests -- problems in describing, explaining, and justifying the mathematical rules they know how to apply correctly. For example, why do you "add decimal places" when multiplying decimal fractions? Why does "invert the divisor and multiply" work in dividing rational numbers? Why do you change the sign of a term when you move it to the other side of the equation? Du Boulay wanted to determine whether representing such rules as procedures could help overcome these problems.

Du Boulay provides a thorough discussion of the advantages and disadvantages of computer programming as a learning activity. He notes the following advantages: Writing programs made students aware of the details and structure of rules, such as those for finding the common denominator, and for adding and subtracting fractions. It gave students practice in finding suitable problem representations, and it allowed them to test their understanding of difficult and abstract textbook language. For example, "the mapping J on the real numbers defined by J: X --> 1/X" could be defined by the following code:

```
FUNCTION J(X)
RETURN 1/X
END
```

Thinking of the function as an instruction to do something made it more concrete. Thus, programming served as a medium for instruction: In explaining a difficult concept, the teacher could approach it in terms of a programming project. The programming approach was directly useful in producing a computational result, as well as being more comprehensible than static equations.

Du Boulay, however, also noted some difficulties with using computer programming as an instructional activity: Programming is difficult to learn and requires a great deal of time and effort. In fact, the difficulty of programming, and the conceptual problems students have with it, may be comparable to the conceptual problems that programming is supposed to help them overcome. Students that have problems with mathematics, for example, are the most likely to have problems with computer programming. Furthermore, it requires a special effort to establish the connection, in the student's mind, between programming and mathematics or another content area. Just as students can learn mathematics without understanding and without being able to apply it to new problems, they can learn programming as an isolated skill. Then programming becomes just one more thing to learn. To the extent that students have to spend a great deal of time learning features of the programming language and solving language-related problems, this distracts their attention from substantive math and science problems and obscures the relation of programming to the substantive problems.

Du Boulay summarizes the advantages and disadvantages of programming for learning as follows:

1. Programming emphasizes the importance of an explicit language for problem representation BUT can cause frustration because getting a workable solution can take a long time and can drag in many additional and irrelevant difficulties.
2. Certain key concepts are well illustrated (function, algorithm, angle, state, and transformation) BUT it is hard to design programming projects which address students' difficulties at the right level of representation, because there are too many compromises with the language of the computer.
3. Splendid opportunities are available for problem solving, planning, refinement, and generalization of solutions BUT students find many ways to sidestep the teacher's good intentions, such as using trial-and-error hacking rather than systematic analysis of a problem. That is, the pedagogical difficulties of the content domain may re-emerge in the programming domain.

The difficulties that du Boulay describes may be attributable largely to his students' very brief exposure to computer programming. If programming were integrated into the science and mathematics curriculum from kindergarten through high school, students could acquire levels of programming skill suitable for the problems that are encountered at progressively higher levels.

The instructional value of programming noted by du Boulay may be explained in part by Richard Mayer's research on teaching and problem solving. Mayer (1980, 1982) distinguishes between simple problem-solving tasks and complex tasks. Complex tasks require relatively more transfer or generalization, whereas simple tasks can be done by rote. Mayer conducted a series of experiments which showed that diverse instructional methods could improve students' ability to solve the more complex type of problem. All these instructional methods required students to produce a representation of the material to be learned that was different from the representation given to them by the teacher. For example, some of the effective methods were note-taking, explaining in one's own words, working with a concrete model of the system, and translating the concepts and relations from one simple programming language to another.

According to Mayer, "This pattern is consistent with the idea [that] good retention requires recall of [a] specific code, but good transfer requires understanding of conceptual ideas."

I interpret Mayer's results in the following way: Rote learning means learning that is tied to one representational system. Whether the student has memorized a formula, a verbal phrase, or an arithmetic rule, her knowledge is closely bound to one representation. This type of knowledge does not constitute deep understanding and cannot be transferred or generalized appropriately, because the important similarities among complex problems rarely appear at the level of their surface representations. Knowledge which supports effective problem solving, therefore, must be detached from any particular representation. The most direct way to accomplish this is to have the student restate the material in a form different from that in which it was initially presented.

Considering the work of Spain, du Boulay, and Mayer, we can conclude that computer programming can be a useful medium for mathematics and science instruction because it provides a framework in which students can restate important theorems, concepts, and processes. On the other hand, if Mayer was able to get effective transfer using note-taking, concrete models, and verbal restatements, what is the point of using computer programming, especially in view of the problems discussed by du Boulay? Does computer programming offer any special benefits to offset the obvious difficulties?

There is a general argument against teaching computer programming to children which states that learning to program is just too difficult for children. As an alternative, microworld software such as Rocky's Boots is recommended for helping children to develop reasoning and problem solving skills. The microworlds are designed to have relatively simple rules of operation and are easy to learn and use. They provide challenging problem solving environments. However, they are limited to an extremely restricted set of problems. Rocky's Boots simulates logic gates, but is not useful for solving statistics problems. In PASCAL, a student can write programs to simulate logic gates, solve statistics problems, do word processing, write a game, or build a spreadsheet. A high level programming language can be used to solve problems in any topic and of any degree of complexity.

Like algebra or arithmetic, a programming language is a general problem solving tool. It is not specialized to one domain. A programming language -- and <u>only</u> a programming language -- provides the means to describe the solution to any solvable problem in explicit, correct, complete, and executable form.

FEASIBILITY

Children can learn to program, given adequate time, informed teachers, and a motivating context. To become proficient in a programming language takes 8 to 10 years, not 8 to 10 weeks. This should not discourage us, however, because it will take 8 to 10 years of schooling before a student arrives at problems which require the full power of a typical high level programming language. The early stages should focus on elementary ideas (Mills et al., in press), such as strings, numbers, variables, sequencing of statements, loops, and guarded statements. These provide sufficient power to model the science and math taught in elementary school.

Teachers are learning to program. Schools are now teaching their teachers. Support comes from legislation such as the Education for Economic Security Act of 1984, which designates funds for developing teacher proficiency in science and math, including computer science. Additionally, new teachers are graduating with skills in programming, due to increasing emphasis on computer-related skills in schools of education. The potential for having a well-informed teaching population in a few years is there.

The present math and science curriculum provides sufficient material to motivate students in learning programming skills at the pre-college level. The goal is not to teach programming for its own sake or as an isolated skill. Just the opposite: The motivation for learning to program should be that programming is a useful problem-solving and learning tool throughout the math and science curriculum.

The educational community should now examine the entire science and mathematics sequence with the purpose of strengthening and supporting it using computer programming. We should determine what minimum programming skills are necessary at each grade level to solve problems and to represent theorems and processes of the material being taught. There is sufficient experience and expertise available in the educational community to develop a workable plan for incorporating programming into the math and science curriculum. The educational benefits of doing this suggest that it is well worth the effort.

References

American Biology Teacher. (Jan., 1985). Vol. 47, Special issue: Software development in Biology education.

Brumby, M. (1979). Problems in learning the concept of natural selection. Journal of Biological Education, 13, 119-122.

Carrier, C., Post, T.R., Heck, W. (1985). Using microcomputers with fourth grade students to reinforce arithmetic skills. Journal for Research in Mathematics Education, 16, 45-51.

Champagne, A., Klopfer, L.E., Gunstone, R.F. (1982). Cognitive research and the design of science instruction. Educational Psychologist, 17, 31-53.

Dalby,J., Tournaire, F., & Linn, M.C. (1984). Making programming instruction cognitively demanding: An intervention study. In D.M. Kurland (Ed.), Developmental studies of computer programming skills (Tech. Rep. No. 29). New York: Bank Street College of Education.

Du Boulay, J.B.H. (1980). Teaching teachers mathematics through programming. Int. J. Math. Educ. Sci. Technol., 11, 347-360.

Feurzeig, W., & White, B. (1984). An articulate instructional system for teaching arithmetic procedures. Cambridge, MA: Bolt Beranek and Newman, Inc.

Glaser, R. (1984). Education and thinking: The role of knowledge. American Psychologist, 39, 93-104.

Kilpatrick, J., & Wilson, J.W. (1984). Taking mathematics teaching seriously: Reflections on a teacher shortage. In: J. Taylor (Ed.), Teacher shortage in science and mathematics: Myths, realities, and research. Washington, D.C.: National Institute of Education.

Larkin, J. (1983). A general knowledge structure for learning or teaching science. In A. Wilkinson (Ed.), Classroom computers and cognitive science. New York: Academic Press.

Lu, C. (1985). The biologist's toolbox: Microcomputers in Biology -- hardware and software profile. BioScience, 35, 178-182.

Mayer, R.E. (1982). Diagnosis and remediation of computer programming skill for creative problem solving, Vol. I.: Description of research methods and results. Final Report, Grant NIE-G-80-0118. Santa Barbara, CA: University of California.

Mayer, R.E. (1980). Contributions of cognitive science and related research on learning to the design of computer literacy curricula (Tech. Rep. No. 81-1). Series in Learning and Cognition. Department of Psychology, University of California, Santa Barbara.

Mills, H., Basili, V.R., Gannon, J.D., Hamlet, R.G., Shneiderman, B., & Kohl, J.E. (In press). The calculus of computer programming. Boston, MA: Allyn & Bacon.

Nickerson, R.S., Perkins, D.N., and Smith, E.E. (1984). Teaching thinking. Report No. 5575. Cambridge, MA: Bolt Beranek and Newman Inc.

Orlansky, J. (1983). Effectiveness of CAI: A new finding. Electronic Learning, 58-60.

Pea, R.D. (1984). What will it take to learn thinking skills through computer programming? In D.M. Kurland (Ed.), Developmental studies of computer programming skills (Tech.Rep. No. 29). New York: Bank Street College of Education.

Pea, R.D., Kurland, D.M., and Hawkins, J. (1984). Logo and the development of thinking skills. New York: Bank Street College of Education. Pea, R.D.,& Kurland, M.D. (In press). On the cognitive effects of learning computer programming. In New ideas in Psychology. Pergamon Press

Smith, C.(1985). Density and weight (Tech. Rep., Nov. 1984). Cambridge, MA: Educational Technology Center, HarvardGraduate School of Education.

Spain, J.D. (1982). BASIC microcomputer models in Biology. Reading, MA: Addison-Wesley.

Sparrow, G. (1985). VisiCalc (TM) for the teacher. Journal of Chemical Education, 62, 139-143.

Stein, J.S., & Linn, M.C. (in press). Capitalizing on computer-based interactive feedback: An investigation of Rocky's Boots. In M. Chen & W. Paisley (Eds.), Children and microcomputers: Formative studies. Beverly Hills: Sage.

Wiser, M. (1985). Heat and temperature (Tech. Rep., Nov. 1984). Cambridge, MA: Educational Technology Center, Harvard Graduate School of Education.

Zeisler, M.J. (1985). Keyboard chemistry. The Science Teacher, 52, 36-40.

The views expressed above are those of the author and do not necessarily reflect policies of OERI, the Department of Education, or the United States Government.