AN ABSTRACT OF THE THESIS OF

Aqeel M. Ahmed for the degree of Doctor of Philosophy in Science Education presented on July 31, 1992.

Title: Students' Thought Processes While Engaged in Computer Programming

Abstract approved: Redacted for Privacy

Margaret L. Niess

The purpose of this qualitative study was to investigate the thought processes of secondary level novice programmers engaged in computer programming for the purpose of generating hypotheses for consideration in future research on the relationship between computer programming and problem solving. A high school BASIC programming course with 14 students from a single school in the tenth through the twelfth grades was selected for the sample.

Data describing students' thought processes while programming were collected during double periods in the 11th and 16th weeks of the fall semester. Students worked in role-assigned partnerships, wherein one student was the problem solver and the other was the recorder. The problem solver's task was to solve the problem using a "think aloud" strategy, while the recorder took notes describing the problem solver's actions to assure that audiotape recordings of the problem solver's voice were maintained.

Following the solution of one problem, these roles were switched.

Analysis of novice programmers' thought processes revealed two categories of student problem solution strategies: coded thinking and debugging. In the coded thinking strategy, students approached the problems primarily from the perspective of BASIC codes. This strategy was similar in nature to activities involved in verbal association learning, a low level thinking strategy identified by Gagné (1970). Students relied on two techniques for debugging syntax and logic errors. They applied a guess-and-check technique to correct syntax errors or asked the teacher for assistance. Similarly, when logic errors were revealed, the subjects typically asked the teacher for assistance and then used the guess-and-check technique to correct the errors. Both techniques utilized lower level thought processes than that required for problem solving learning. Analysis of the subject programming processes revealed that problem solving processes, as identified by Polya (1988), were not involved. Future research should examine students thought processes when working with a compiled language such as Pascal. In addition, future research should investigate the thought processes of students who have had more experience than a single term of programming. A case study of from two to three students explored over a longer period of time may provide a clearer description of student thought processes.

Students' Thought Processes While Engaged in Computer Programming

by

Aqeel M. Ahmed

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed July 31, 1992

Commencement June 1993

APPROVED:

Redacted for Privacy

Associate Professor of Mathematics Education in charge of major

Redacted for Privacy

Chair of Department of Science & Mathematics Education

Redacted for Privacy

Dean of Graduate School

Date thesis is presented July 31, 1992

Typed by A. Ahmed for Ageel M. Ahmed

Acknowledgment

Four years of Oregon State University schooling has equipped me with a strong background and experience in science education. Moreover, studying at Oregon State University has broadened my views of life and strengthened my teaching abilities.

My work could not be accomplished without the instruction, assistance, kindness, and patience of Dr. Margaret L. Niess, my major professor. She has always encouraged me to think and to learn for myself. I could not reach this level of education without her invaluable advice and encouragement. I would like thank Dr. Norman G. Lederman for his help and advice. Dr. Lederman was always available for research help, specifically in his doctoral seminar. I would also like to thank the members of the committee who helped, encouraged and stimulated me along the way: Drs. Jake Nice, Toshimi Minoura, and Kenneth Johnson. I have learned a lot through discussions as well as from the several meetings with my committee.

I thank all my doctoral colleagues whom I spent time in learning and friendship. My friend Mark Latz was a great help in many ways. I enjoyed working with him in conducting research, gardening, and playing racket ball. I

thank Kathy Howell for reading my dissertation and providing suggestions.

I need to thank the school, the teachers' and the students who agreed to participate in this study. Without their cooperation and patience, the objectives of my research could not have been achieved.

I need to thank the University of Bahrain for their financial support during my study. My thanks goes to Dr. Abdulla Al-Hawaj, the chairman of the Computer Science Department at the University of Bahrain, for his support and advice during my period of study in the U.S.

I would like to show my appreciation to my wonderful wife, Kelly, and my lovely daughters, Shadia and Nadia. Nadia was born just one week after my defense. With them I have a nice and happy life. I would also like to thank my parents, my brothers, and my sister for their encouragement and support.

Table of Contents

Table of Contents (continued)

List of Tables

Students' Thought Processes While Engaged in Computer Programming

CHAPTER I

THE PROBLEM

Introduction

Problem solving skills help students undertake problems in a practical and effective manner; yet, prior knowledge and experience does not always prepare students to deal with new and unexpected situations. For mathematics learning, the development of problem solving skills is widely recognized as one of the major goals of the instructional process. Problem solving skills and applications are two of ten basic skills identified in recommendations provided by the National Council of Supervisors of Mathematics (1978). Moreover, a task force of the National Council of Teachers of Mathematics (NCTM, 1989), given the mission of determining curriculum guidelines for the 1990s, selected problem solving as an area of primary concern for mathematics education curricula. Thus, it is generally accepted that problem solving is a fundamental aim of mathematics instruction at all levels.

While stating the essential value of the problem solving process, it was nonetheless true that the results of the second mathematics assessment of the National Assessment of Educational Progress (NAEP) indicated that students achieved poorly on exercises at the application or problem solving levels (Carpenter, Corbit, Kepner, Lindquist, & Reys, 1980). It has been posited that the application of computer technologies provides one means to respond to this problem. Hence, the literature of computer technology provides numerous claims that learning computer programming skills will contribute to significant cognitive development. Specifically, Papert (1980) suggested that learning computer programming skills involves techniques that serve to develop problem solving skills. Based upon the commonly stated goal of teaching problem solving, instruction in computer programming in the elementary and secondary schools has become a popular part of school curricula.

Feuzeig, Horwitz, and Nickerson (1981) argued that learning programming skills provides the opportunity to develop rigorous thinking, to learn to use heuristic, and to nourish self-consciousness about the process of problem solving. Similarly, Linn (1985) analyzed the cognitive requirements of different levels of programming, such as precision and structural organization, in the belief that these requirements were transferred to the general problem solving process.

Since computer programming has often been characterized as a type of problem solving, it may be possible that learning programming skills has a transfer effect for general problem solving. The steps used to design programs are similar to the steps developed in the problem solving process (Kransor & Mitterer, 1984). Polya (1988) identified four basic steps used in a wide variety of problem situations through which problem solvers are to proceed: 1) understanding or analyzing the problem, or attempting to understand its specific aspects and requirements for an acceptable solution; 2) deciding on a plan or, based on the problem solver's skills, selecting a method or strategy to solve the problem; 3) carrying out the plan, or the mechanical process of problem solution; and 4) looking back, or review and reconsideration of the proposed solution to assure that it is complete and correct.

Similarly, Pea and Kurland (1983) described four general steps taken by programmers in the creation of a computer program: 1) understanding the problem, or comprehension of the programming task accompanied by the ability to represent the problem; 2) designing and planning, or defining the overall flow of the program and identifying the procedures that can be used; 3) coding the program, or writing and entering program codes into the computer; and 4) debugging the program, or using various strategies to modify the program to the end of generating the desired outcome.

Given some degree of relationship between computer programming and problem solving skills, the transfer effect between two learning systems, according to Thorndike is dependent upon the presence of identical elements in the original learning method and the new learning method that the former serves to facilitate (Hergenhahn, 1982). Blume and Schoen (1988) have stated that creating a program and solving nonroutine mathematical problems have certain broad similarities in the areas of (a) analyzing a problem situation and planning for its solution, (b) selecting and applying a solution strategy, and (c) checking or verifying the completed program or solution strategy. These elements may be either substantive or procedural identities (Higard & Bower, 1975). Therefore, transfer may occur under circumstances where learners have practiced programming skills in several situations and are aware of the general advantages of these techniques. Moreover, it follows that transfer may occur from learning programming skills to the development of problem solving skills.

However, in point of fact little research has been conducted on what students really learn when they are programming. In addition, research is needed that would examine the relationship between computer programming and student cognition, addressing the nature of student thought processes as they learn to program on the computer (Pintrich, Berger, & Stemmer, 1987).

Statement of the Problem

The purpose of this study is to investigate the thought processes of secondary level novice programmers engaged in computer programming by addressing the following questions:

1) Do students' thought processes change as a result of additional instruction and experience?
2) What are students thinking while actively engaged in program development?
3) Do students' thought processes resemble the thought processes involved in generic problem solving (i.e., planning procedures) while they are actively engaged in program development?
4) Are students' thoughts a reflection of scientific programming codes (i.e., language syntax) or of the problem that is posed?

Descriptive data of students' thought processes during programming procedures are needed to form a basis for the generation of hypotheses or patterns in the relationship between computer programming and problem solving skills, as well as to provide a detailed description of the nature of students' thinking processes while actively engaged in programming. From the hypotheses developed, further studies can be conducted to clarify the transfer relationship between programming and problem solving skills.

Significance of the Study

At all levels, educational institutions are including computer programming courses in their curriculum in spite of the conflicting evidence supporting claims made about the value of learning to program. Pea and Kurland (1984) concluded that little empirical evidence exists to support the claims that programming enhances students' thinking or problem solving skills. Perhaps the reason for this lack of encouraging results is that the studies that have been conducted have not been based on a solid theoretical framework. Specifically, with respect to the relationship between programming language instruction and problem solving skills, many of the previous studies have not been sufficiently grounded in problem solving theory as presented in an appropriate theoretical framework (Palumbo, 1990). A number of researchers have jumped directly to a study of the relationship between programming and students' cognition processes prior to gathering basic information on the nature of the thought processing procedures involved in computer programming. Thus, given the inability to validate the general claim for a transfer relationship between programming instruction and human cognition, research is needed to study students' thinking patterns as they engage in computer programming and to generate hypotheses for further research concerning the link between learning computer programming and problem solving skills.

CHAPTER II

REVIEW OF THE LITERATURE

Introduction

In a large, nationwide representative sampling conducted in American school systems, Becker (1983) found that 47 percent of the elementary schools and 76 percent of secondary schools offer at least one course in computer programming. In fact, the teaching of computer programming in elementary and secondary schools has become a popular part of the curriculum and one of its principal program goals is that this instructional form encourages the development of student problem solving skills. Papert (1980) claimed that such programming environments as *Logo* provide experience that may reduce the time between student concrete and formal ability stages of cognitive development. Therefore, he suggested that the experience of planning, executing, and debugging programs in a computer-rich unstructured environment may help students make the transition to formal systematic reasoning. As noted in the previous chapter, this position has been reinforced by other studies, a number of which have hypothesized that there is a transfer effect between the disciplined approach, rigorous thinking, and need for precision required

in computer programming and equivalent skills required for successful problem solving (Feuzeig et al., 1981; Linn, 1985).

Therefore, this chapter includes a review and an analysis of the empirical research which has been presented on the effect of learning computer programming upon the development of human cognition processes. Problem solving is defined and alternative problem solving processes are discussed in the first section, accompanied by consideration of research supporting instruction in the Polya (1988) method at all levels of public education. A second section reviews studies that have investigated the relationship between programming and general cognitive outcomes.

Problem Solving

A problem is a situation, quantitative or otherwise, which an individual or a group of individuals confronts that requires resolution, and for which a readily apparent solution path does not exist (Krulik & Rudnick, 1988). In this sense, to exist, a problem must satisfy the three criteria of acceptance, blockage, and exploration. Acceptance involves motivation to become involved in the solution of a problem; blockage implies that the problem solver's initial attempts to solve a problem do not work;

and exploration encompasses the exploration of new methods of problem attack.

According to Wickelgren (1974), all problems require three types of information: information concerning givens, information concerning operations, and information concerning goals. The givens refer to a set of expressions that the problem solver recognizes or accepts as valid for the specific problem in the initial stage of problem solution. Operations refer to actions that the problem solver performs on the given conditions or expressions to generate a desired goal. Goals refers to the measure of a terminal expression desired within the problem solution. Consequently, problem solvers must accept the challenge of the problem, otherwise the problem cannot be considered a problem for that person. Engagement in the problem solving process requires that the solver utilize previously acquired knowledge, skills, and understanding to fulfill the demands of unfamiliar situations (Krulik & Rudnick, 1988).

The problem solving process begins with consideration of the initial problem conformation and terminates when a solution has been obtained and has been considered valid with regard to the initial situation. In general, the process is a heuristic technique for solving problems for which a correct solution cannot be guaranteed, serving as guidelines for attempts to solve the problem. Rubinsten (1975) described the problem solving process in a simple four-stage model:

Stage 1, Preparation: Problem solver examines each element within the problem and studies the relationship between these elements;

Stage 2, Incubation: Problem solver thinks about the problem, confronting frustrations inherent in the inability to identify a solution for the problem;

Stage 3, Inspiration: Problem solver obtains a clue to the means to solve the problem; and

Stage 4, Verification: Problem solver checks the inspiration against the desired goal.

The Rubinstein (1975) model was based on the experience of scientists who had drawn upon inspiration to solve difficult problems. The four stages in the model take place either in parallel or in series, dependent upon the problem situation. Typical strategies used to approach a problem involve procedures from the initial stage to the desired problem goal through a set of operations with no distinctly defined intervening stages (Newell & Simon, 1972). Some proofs for mathematical theorems and some riddles are problems of this type.

Dahmus (1970) developed a problem solving method that suggested a transational process in which each verbal statement is translated into a corresponding mathematical statement. In this method, emphasis is grounded within the particular problem, operationalizing a procedure that is similar in construction to a jig-saw puzzle, or built

piece-by-piece without the necessity of considering the whole.

The mathematician Polya, in his study *How to Solve It* (1988),[1] identified four basic steps of the problem solving process: understanding the problem, making a plan, carrying out the plan, and looking back. To understand the problem, the problem solver must identify the meaning of the key words, determine the parameters for relevant data, search for relationships among the data, and then understand what is being asked. Frequently, the problem solver must think in terms of specific questions to even begin to understand the problem, that is: What is known? What are the data? What is the condition? The problem solver must try to separate the parts of the condition to determine the relationships between the data and the unknown. Moreover, the problem solver may obtain clues to the solution of the problem by thinking of similar or familiar problems that have the same or similar unknowns. In addition, the problem solver may need to think of other data that will be useful to determination of the unknown, and may need to try to change the unknown(s) or the data, or both as necessary, so that the new data and the new known facts are closer to one another.

After understanding the problem, the following step is to devise a plan for solution (Polya, 1988). In this step

---

[1]Published originally in 1957, the reference is the second edition.

the problem solver may need to review similar problems which were solved previously. If the problem solver finds similar problems or is able to solve simpler problems, then a basic condition of the process has been met. Once a related problem is identified, the problem solver can apply the method used to solve the related problem to the solution of the problem at hand. To devise a plan, the problem solver must understand the problem as a whole, examining and gaining thorough understanding of the relation of its principal parts. The problem may require separation into its logical component parts, each to be considered separately. Following this process, the problem solver may continue to decompose the problem, recombining its elements in a new manner as necessary.

In the third step, carrying out the plan, the problem solver checks each step of the plan to determine if the strategies devised function correctly (Polya, 1988). The problem solver must give consideration to the order of the plan without the omission of any of its details or losing sight of the connections between its principal steps. Thus, the problem solver should proceed in an ordered manner. For the fourth step, looking back, the problem solver checks the results to determine whether the outcomes are reasonable and correct. If possible, the problem solver extends and generalizes the solution to a similar situation by finding or devising and then exploring a related problem based upon the problem solution.

The results of several studies have supported the Polya (1988) method for teaching students a logical problem solving process. Gordan (1977) sought to determine if there were certain aspects of cognitive/personality styles that affected prospective teachers' choices for a problem solving paradigm. Fifty-six undergraduate elementary education majors were tested on the conceptual level with respect to field-dependence-independence. The subjects were given a self-reporting instrument for the measurement of their choice of problem solving methods and presentation formats. The results indicated that prospective teachers who had high conceptual levels tended to choose the Polya method, whereas prospective teachers with relatively low conceptual level scores tended to choose the Dahmus (1970) method.

Bassler, Beers, and Richardson (1972) assessed the relative effects of two distinct strategies of instructions for the solution of verbal problems, comparing the methodologies developed by Polya (1988) and Dahmus (1970). For seven days, 53 ninth-grade algebra students were given instruction in either method and then administered a post-test and a retention test for subsequent analysis of equation criterion and problem solution criterion. The results indicated that the subjects who selected the Polya method for their instruction scored higher than those who selected the Dahmus method for the equation criterion, but that there were no significant differences between the two

groups for the problem solving criterion. More recently, Rosati (1985) conducted a study in which computer problem solving tutorial routines were written according to the Polya strategy for the use in an engineering statics course. The work of a pilot group of students was monitored, with the result that the routines were recommended as a course supplement, but not in the sense of serving as a replacement for traditional means of instruction.

Despite these less than clear research results, the Polya (1988) method of problem solution has permeated throughout educational institutions at all levels. Several educators have advocated adaptations of the Polya steps as a sound instructional sequence for problem solving instruction. LeBlanc (1982) stated that the most important skills a teacher can offer in elementary school mathematics are the problem solving skills identified by Polya. Red (1981) noted that the Polya method contains some of the necessary elements that engineers use for problem solution, and, as a means to alleviate an apparent decline in student problem solving ability, adopted the Polya approach.

The discussion of the use of heuristic processes in problem solving by Polya (1988) has served as a framework for the NAEP assessment procedures, serving to provide useful insights into student problem solving abilities (Carpenter et al., 1980). The NCTM (1989) has stated that problem solving is the essential focus of the mathematics curriculum.

Relationships Between Programming and General Cognitive Outcomes

This section is presented in three parts, including studies that have investigated the relationship between programming and general cognitive outcomes, studies that have analyzed the relationship between learning computer programming and problem solving, and studies that have examined the relationship between learning computer programming and specific cognitive skills.

General Cognitive Outcomes

Clements and Gullo (1984) compared the effects of learning Logo programming to computer-assisted instruction (CAI) procedures. The subjects were 18 first-grade students, with an average age of 6 years and 11 months, from a Midwestern middle school class, each of whom were pretested for receptive vocabulary and reflective and divergent thinking, using the Matching Familiar Figures Test (MFFT) and the Peabody Picture Vocabulary Test (PPVT). In addition, an instrument designed by Markman (1977) was used to assess the subjects' ability to monitor and evaluate their own cognitive processes (i.e., metacognition) and the McCarthy Screening Test (MST) was used to measure cognitive development.

Subjects were then randomly assigned to either a CAI group or a Logo programming group and given a sequence of 45-minute learning sessions for two days a week over a 12-week period (Clements & Gullo, 1984). The CAI sessions consisted of computer-based lessons that were concentrated upon reading and arithmetic concepts. The Logo sessions consisted of a sequenced programming instruction. At the end of the training period, subjects were given posttests for cognitive style (reflective and divergent thinking), metacognition, cognitive development (operational competence and general cognitive measures), and the ability to describe directions. Statistical t-test analysis revealed significant differences in metacognitive ability and the ability to describe directions between the Logo group and the CAI group. It was concluded that the subjects given Logo programming instruction reflected greater achievement than the CAI students for these two measures.

Clements (1986) conducted an assessment of the effects of learning computer programming skills and CAI on cognitive skills (i.e., classification and seriation operations), metacognitive skills, creativity, and achievement (i.e., reading, mathematics, and the ability to describe directions). The subjects were 72 children randomly selected from a middle school, including 36 first-grade (18 girls and 18 boys, mean age 6 years, 10 months) and 36 third-grade children (19 girls and 17 boys, mean age 8 years, 10 months). All subjects were randomly assigned to

one of the three treatments: CAI, Logo programming, or a control group.

At the beginning of the year, all of the subjects were pretested for operational competence, creativity, reading achievement, and mathematics achievement (Clement, 1986). The MFFT test was used to measure cognitive skills, two tasks designed by Markman (1977) were used to assess the subject metacognition, and the Torrance Test of Creative Thinking was administered to measure the ability of the child to think divergently in a nonverbal mode. Subjects were also given a street map, asked to draw a path from their house to a certain store, and then they were asked to describe their directions. The, for a period of 22 weeks, sequences of 44 sessions in either Logo (*Terrapin*) or CAI were presented. For the Logo programming treatment, lessons were concentrated on programming concepts, whereas for the CAI treatments, subjects used computer programs to learn problem solving skills, arithmetic, and reading. The control group simply participated in regular schedule classroom lessons. At the end of the period of study, all subjects were administered posttests to assess their cognitive skills, metacognitive skills, creativity, and achievement.

A statistical analysis of variance (ANOVA) procedure of the results indicated there were significant differences for cognitive skills, metacognitive skills, and direction descriptions between the groups in favor of the Logo group

with respect to both the CAI and control groups (Clement, 1986). However, the ANOVA revealed there were no significant differences among the three groups for reading and mathematic achievements.

Turner and Land (1986) investigated the effects of instruction in Logo skills upon the comprehension of specific mathematical concepts and levels of cognitive development, based upon the following hypotheses: 1) By learning Logo, students will achieve higher mathematics concepts and attain higher levels of cognition than students who do not learn Logo; and 2) students who learn a greater number of Logo skills will achieve significantly higher for mathematics concepts and attain a higher levels of cognition than students who learn minimal Logo skills.

The sample consisted of 181 subjects from seven classes in four inner-city schools in the Midwest (Turner & Land, 1986). The experimental group consisted of 91 subjects who studied Logo for one hour each week for 16 weeks as part of their mathematics curriculum. The experimental group consisted of 59 fifth- and sixth-grade subjects from a magnet school and 32 sixth- through eighth- grade subjects from regular schools. The control group consisted of 90 sixth through eighth grade subjects, representing all ability levels, who studied mathematics by traditional methods for 16 weeks. All subjects were pretested for selected mathematics concepts and cognitive development. Following administration of the instructional methods, sub-

jects in the Logo group were given a posttest developed by the researchers, including 22 multiple choice questions directed at the assessment of mathematics concepts, and a Social Science Piagetian Inventory (SSPI) was implemented for all subjects to measure cognitive development. Results from a one-way analysis of covariance indicated there were no significant differences between the Logo group and the control group for mathematics achievements or for cognitive development. It was thus concluded that learning Logo programming skills does not have an effect upon student cognitive development or mathematics achievement abilities.

Linn and Dalbey (1985) studied the cognitive consequences of programming instruction in BASIC to assess its effects with respect to general ability, access to computers outside the school, previous computer experience, and interest shown in computers. Selected schools from within 50 miles of the Lawrence Hall of Science at the University of California (Berkeley), which offered rigorous computer programming courses, were surveyed. Requirements were that programming courses had to be at least 12 weeks in duration, with a minimum of eight computers available for students and for teachers, each of whom were required to have at least 100 hours of programming experience. For subjects from the selected schools, the Advanced Progressive Matrices, Set I (Raven, 1965), was used to measure subject abilities, following which subjects were placed in low, medium, and high ability groups according to their Raven

test scores. A second test (Headlines) was implemented to assess interest in computers, science, and nontechnical areas. A Final Programming Assessment test, with sections on comprehension, reformulation, and design, was applied to assess programming skills.

A significant correlation coefficient was found between the means for the ability groups and programming performance at typical sites (Linn & Dalbey, 1985). Thus, it was determined that there was a relationship between the form of instruction, access to computers, and subjects for the outcomes of programming instruction. The researchers concluded that exemplary instruction moves students further along the chain of cognitive accomplishments than courses which offer typical patterns of instruction.

Howell, Scott, and Diamond (1987) studied the effect of Logo instruction on the cognitive development of children from five to seven years old. Cognitive development was measured using the Piagetian tests, administered as both pretests and posttests to measure the conservation of length, measurement, and the ability to identify Euclidean shapes. The study was conducted over a period of six months in individual daily sessions of 15-20 minutes. The control group consisted of 40 kindergarten students who received the regular daily lesson, whereas the treatment group consisted of 40 kindergarten students who learned Logo programming for approximately 75 to 80 minutes per week. Chi-square analyses of the posttest results on the

conservation of numbers and conservation of length showed no significant differences between the two groups, and it was concluded that learning Logo skills did not accelerate the cognitive development of children from five to seven years of age.

Gallini (1984) compared the effects of Logo programming and CAI for two types of cognitive outcomes: the ability to execute directions and the ability to formulate directions. Subjects included 44 fourth-grade students from urban school districts in the southeastern United States, one-half of whom were assigned to a Logo group and one-half to a CAI group. The subjects reflected a variance in achievement levels, based upon teachers' ratings and the results of standardized achievement test scores (i.e., a statewide basic skills test).

Each treatment was administered in 25-minute periods, three times each week for five weeks (Gallini, 1987). Subjects in the CAI treatment group received instruction in the educational computer software, *Koala Pad* and *Rocky's Boots*, concentrating upon flowcharting skills and programming activities. The programming activities required subjects to identify tasks, break the tasks into smaller parts, and to write a program to instruct the computer to execute each task. The subjects in the Logo treatment group learned Logo programming. Following each session, subjects were given activities that required writing a Logo program to draw different geometrical figures. Subjects

were given pretests and posttests based on a ten-item scale concentrated upon the tasks of following and formulating directions. Statistical analysis indicted that the subjects who learned Logo scored significantly higher than the CAI subjects in the formulation of directions.

Hunter, Kemp, and Hyslop (1987) investigated whether a curriculum utilizing Logo would support greater cognitive growth than a traditional curriculum. Subjects were selected from six elementary schools within the area of the Calgary Board of Education. The selection of schools was based upon staff interest and the availability of computer and teaching staff with knowledge of Logo. Grades three and five were selected from these schools since students from this age group were considered to be sufficiently mature to undergo group and individual testing. The study, lasting six months, consisted of three groups: learning problem solving with Logo, learning Logo in a traditional manner, and a control group.

The Logo group learning problem solving consisted of 76 subjects from the three of the schools (Hunter et al., 1987). For this group, teachers were given inservice training in Logo programming, based upon the Feurestein (1981) concept of mediated learning, to facilitate the teaching of problem solving. A second group of 78 subjects from two of the schools learned Logo in the traditional manner. For this group, teachers were given an outline of a specific Logo concept that was to be covered during the

treatment period. The control group consisted of 23 students from a class at the remaining school. For this group, the teacher used the types of problem solving activities that were normally carried out as a part of the curriculum (i.e., skill development which did not involve computer programming). All subjects were given pretest and posttest cognitive ability measurement tests. From the test results, it was determined that the subjects who learned Logo in the traditional manner scored significantly higher for the non-verbal test items than did either the subjects learning problem solving with Logo or the control group. No other significant differences were determined.

Problem Solving

Swan and Black (1988) examined relationships between learning Logo programming skills and the development of problem solving skills, based upon five problem solving strategies: forward chaining, backward chaining, systematic trial and error, alternative problem representation, and analogical reasoning. Three hypotheses were tested, including: 1) When problem strategies were applied in the Logo environment, they would be transferred to non-computer domains; 2) Subjects would reflect developmental differences in the ability to acquire and transfer problem solving skills; and 3) Subjects would reflect different abilities in transferring problem solving skills, dependent upon the base context(s) in which these skills were acquired.

The subjects for this study were 133 students in the fourth through eighth grades of a private suburban elementary school a minimum of 30 hours of previous instruction and experience in Logo programming (Swan and Black, 1988). All subjects were given separately designed pretest and posttest measures of their ability to solve problems by implementing each of the six problem strategies. Subjects were randomly assigned by grade to one of the three contextual groups to receive, respectively, graphics, list, or combined graphics and list problems, based upon application of a consistent instructional sequence for each strategy. The subjects were introduced to each problem solving strategy through group activities designed to provide concrete off-computer models of the cognitive processes involved in each strategy, and worked on problems during two 45-minute class periods per week for approximately 12 weeks. From the test results, significant differences were found between pretest and posttest scores for all of the strategies except backward chaining. It was thus concluded there was a positive relationship between learning Logo programming skills and all of the remaining problem solving strategies.

Swan (1989) investigated the relationship between Logo and learning problem solving based upon two research questions: 1) Is practice in particular problem solving strategies superior to discovery learning supporting the acquisition and transfer of problem solving strategies within

Logo programming environments; and 2) Is the Logo programming environment particularly supportive of the acquisition and transfer of problem solving skills? The study focused on implementing the five problem solving strategies of subgoal formation, forward chaining, systematic trial and error, alternative representation, and analogy. Subjects included 100 students from the fourth through sixth grades of a private suburban elementary school, each of whom had at least one year of prior experience in Logo programming.

The subjects were given pretest and posttest measures based upon their ability to solve problems requiring the use of each of the five problem solving strategies (Swan, 1989). Two different versions of each test were used and were randomly assigned by condition for the pretest. The subjects were then given the alternative forms of each test for the posttest. All subjects worked in pairs during their regular computer classes for two 45-minute periods each week for a period of approximately two and one-half months. Each subject was randomly assigned by grade to one of three treatment conditions, either a Logo graphic condition, a discovery learning condition, or a Logo project condition. For the first two conditions, subjects addressed the same basic problem solving instruction procedures, but differed in the practice environment; subjects in the Logo graphics group received practice problems involving Logo graphic programming, while students in the second group worked on a similar problem using cut-

paper manipulation. Subjects in the Logo project group learned Logo programming, but were not instructed in problem solving.

For the subgoal formation strategy test, students were measured on their ability to solve mathematical word problems, breaking them into parts as well as seeking solutions. Subjects were tested for forward chaining skills by using a paper and pencil version of the computer program *Rocky's Boots*. In the systematic trial and error strategy test, subjects were tested with different symbol combinations to attain coherent decoding systems. In addition, two decoding exercises were used to test the subject abilities to systematically utilize trial and error strategies. From the test results, significant differences among groups supported the hypothesis that learning Logo programming provides students with improved problem solving skills.

McGrath (1988) examined the relationships between the transfer of problem solving and groups of high school students who learned either a first or a second programming language, or who received no programming instruction. Six classrooms were selected for this study, five from two medium-sized towns in Illinois and from a small town in Wyoming. The study was composed of five groups for a treatment period which lasted for one year (i.e., two semesters). Groups one and two were composed of BASIC classes in which the subjects learned their first programming language. However, in each class a few of the sub-

jects had taken Pascal as their first language on an independent study basis. Group one consisted of 21 subjects, 19 of whom learned BASIC and two of whom learned Pascal; group two consisted of 20 subjects, 15 of whom learned BASIC and five of whom learned Pascal. Groups three and four consisted of two Pascal classes for subjects learning their second programming language. Group three consisted of 29 subjects, 20 of whom learned Pascal and nine of whom 9 learned BASIC. Group five consisted of 23 students who did not learn a programming language.

All the subjects were pretested during the third week of the first semester (McGrath, 1988). During the second semester, groups two and four were given a two-week problem solving intervention, whereas the remaining groups held their normal classes. In the two intervention groups, the teachers talked explicitly about programming problem solving. Creative problems were assigned each day, and solutions and tactics were discussed. Three pieces of published software were implemented during the two weeks of intervention: *Where in the World is Carmen Sandiego*, the MECC problem solving program, and *Rocky's Boots*. All subjects were given a posttest at the end of the second semester. The testing results revealed there was a significant difference in favor of subjects receiving the intervention. Accordingly, it was determined that subjects receiving the problem solving intervention during their first programming language course improved the use of the biconditional and

in creative problem solving. In addition, subjects learning their second language significantly outperformed those learning a first language in the debugging procedures.

Blume and Schoen (1988) investigated the problem solving process using eighth-grade programmers and nonprogrammers as subjects, based upon the hypothesis that the programming subjects would use more techniques, including variables and equations, than the nonprogrammers, and would make increased use of systematic approaches as well as planned or prepared processes. Six classes were selected from a midwestern junior high school, three each from fall and spring terms. There were 58 subjects in the three fall classes and 33 subjects in the three spring classes.

The Iowa Test of Basic Skills (ITBS) problem solving test was administered in October prior to the conduct of the courses and was used to assess prior problem solving achievements. Three instruments were used for this study, including interview problems, written word problems, and written logic tests. Specifically, 12 problems were designed to measure subject abilities in problem solving techniques, five of which were selected for use because they elicited a wide range of problem solving techniques. The ITBS was implemented to measure students' abilities to understand word problems and to apply specific problem solving heuristic. In addition, 10 multiple-choice items and eight open-ended items were developed to measure sub-

ject abilities to apply the sequential logic inherent within computer programming.

Subjects enrolled in BASIC classes for the fall term were interviewed during the last week of the term, while students who enrolled in BASIC for the spring term were interviewed during the first week of the beginning of BASIC programming course (McGrath, 1988). In each interview, subjects were asked to "think aloud" while solving the problem, as the interviewer took notes and audiotaped the lesson. The subjects' written work was also collected. Statistical analyses indicated that programming subjects used systematic trial procedures more frequently than did the nonprogrammers. It was also determined that the programming subjects checked for and corrected more errors to obtain potential solutions. Dalton (1986) compared the effects of the use of Logo with teacher-directed problem solving instruction and conventional mathematic instructions for problem solving abilities, basic skills achievements, and the attitudes of junior high level students. Subjects included 97 students from five seventh-grade mathematics classes. Three instructional treatments were implemented: a problem solving strategy, a structured Logo treatment, and a control using neither technique. Subjects were classified as high, average, or low in prior achievement based on sixth grade CTBS scores. Problem solving treatments consisted of approximately 20 hours of instruction in the strategies of guess and check, make a

table, patterns, make a model, elimination, and simplify. Subjects explored the turtle graphics capabilities of the *Terrapin* Logo language. Independently, each subject completed lessons containing a list of new commands and exercises. The control group was given additional time for completing school assignments and/or recreational reading.

The Program Criterion Reference Test (PCRT), or a test of student mastery of grade level objectives, and the mathematics subtests of the Comprehensive Test of Basic Skills (CTBS) were selected to measure the dependent variable (Dalton, 1986). Students' attitudes were evaluated with the Revised Math Attitude Scale, a Likert-type scale questionnaire, and the School Attitude Measure (SAM). Two measures of problem solving skills from the Test of Cognitive Skills (TCB), consisting of Memory, Analogy, Sequence, and Verbal Reasoning scales, were used. Treatments consisted of 45-minute weekly instruction periods over two months (i.e., 20 sessions in all). At the end of the experimental periods, subjects were posttested for high-level thinking skills and for attitude. Statistical analysis failed to reveal significant differences among the three groups. Accordingly, it was suggested that the problem solving skills fostered through the use of Logo may not have a transfer effect outside of the context of Logo.

Rieber and Lioyed (1986) investigated whether, given the experience of Logo programming, young children would acquire skills in problem solving and basic geometry.

Subjects for both groups were second-grade students from intact classes selected from different school districts, consisting of 25 students (average age = 8.08 years) from regular public schools for an experimental group and 22 students (average age = 7.82 years) from similar schools for a control group. The experimental group was given one hour of Logo programming instruction each week for three months and the subjects were asked to complete a series of Logo activities. The control group received the regular class content. With respect to content, both groups received similar instruction and used identical textbooks.

Problem solving ability and geometric knowledge were the two dependent variable measures (Rieber, 1986). The problem solving measure was based on two classical Piagetian activities used originally as examples of problem solving by individuals at the stage of formal operations. The measure of geometric concepts included recognition of the following parts: angles, concept of angles, line segments, and the rotation of given figures. The analysis of the data indicated significant differences, thus the findings of the study supported the view that successful programming interactions would serve to encourage the development and exercise of problem solving abilities (Papert, 1980).

Horner and Maddux (1985) investigated the effect of Logo programming on locus of control, attitude toward math, and ability to recognize the size of geometric angle prob-

lem solving abilities. The 74 subjects who participated in this study identified themselves as either learning disabled (LD) or non-learning disabled (NLD) students at a junior high school in an urban west Texas school district. The two experimental groups, including 16 LD and 21 NLD subjects, and the two control groups, including 20 LD and 17 NLD subjects, were drawn from an intact group of mixed seventh and eighth-grade LD math students and one intact group of NLD eighth-grade math students. Experimental group subjects received Logo instruction for 14 sessions during their math classes, each lasting 55 minutes; the control group continued with the regular math curriculum.

The results of pretests and posttests were collected using four instruments: the Group Assessment of Logical Thinking (GALT), the Intellectual Achievement Responsibility Questionnaire (IARQ), the Fennema-Sherman Math Attitude Scale, and the Horner Angle Recognition Test (HART). For pretesting, instrument items were read orally to the subjects to avoid the effects of poor reading. Following statistical analysis of the test results, it was determined that there were no significant differences for problem solving, locus of control, math attitudes, or angle recognitions among the experimental and control groups (Horner & Maddux, 1985).

Specific Cognitive Skills

Degelman, Free, Scarlato, Blackburn, and Golden (1986) examined the effects of a short-term single-keystroke Logo experience upon rule-learning tasks among kindergarten children. The sample consisted of a class of 15 kindergarten students from a private day-care center. Eight children (i.e., seven girls, one boy) were randomly assigned to receive the Logo experience, and seven students (i.e., three girls, four boys) were assigned to a wait-list control group. The Logo group students received 15 minutes of instruction each school day for five weeks. The children were free to create any design they wished using Logo. All subjects were given between two and four rule-learning problems to measure logical thinking. After the subjects were tested, the control group initiated a similar period of Logo training. Statistical analysis of the data for the matching groups revealed significant differences between the two groups, and it was concluded that children receiving Logo instruction achieved higher scores in problem solving tasks than children who did not learn Logo.

Kurland, Pea, Clement and Mawby (1986) examined the development of thinking skills and programming ability among high school students, based upon the following questions:

1) Do students who learn programming for two years achieve better in reasoning and thinking skills than students who learn one year of programming?
2) Is there a relationship between programming, math, and reasoning skills; in other words, are certain math and reasoning skills good predictors of the success of the programming course?
3) Are students able to write advanced programs after their second year of programming?

The subjects for the study were drawn from an urban public high school with a mixed ethnic and socioeconomic status population (Kurland et al., 1986). Fifteen teneth through twelfth grade students learning second year programming, and who varied in ability according to their GPA and grade levels, represented the experimental group. The control group consisted of nine students with no prior programming experience and six students who had approximately one year of programming experience. At the beginning of the year, students were given pretests to predict their performances in programming classes. The pretests covered the following concepts: procedural reasoning, planning, and mathematics. The posttests were administered at the end of the year and were assessments of subject improvements in cognitive skills. Statistical analysis indicated there were no significant results, thus it was concluded that even after two years of computer programming study, many students retained only rudimentary programming

skills. The study results also suggested that programming skills do not appear to have a transfer effect to other domains which share analogous formal properties.

Clement, Kurland, Mawby, and Pea (1986) investigated the relationship between analogical reasoning skills and aspects of programming that involved mapping structures across problems. The subjects were 17 ninth and eleventh grade females enrolled in a Logo programming course, offered as a part of a six-week program designed to improve math skills. Subjects spent 90 minutes each day learning Logo programming, for a total of approximately 45 hours of instruction. The analogical reasoning task was adapted to measure subject abilities in the determination of structural similarities between two story problems. Data analysis indicated there was a significant correlation for the subprocedures across programs. Thus, results suggested that analogical mapping ability is related to the practice of identifying and writing subprocedures that can be used in various programming tasks.

McCoy (1988) examined relationships between computer programming experience, mathematics experience, and general variable skills. The subjects were 46 summer computer camp students (ages 9 to 17 years) with varied ability levels in both computer programming and mathematics. A programming placement test was performed to measure subject knowledge of computer programming and mathematics experience was defined as the number of successfully completed years of

higher mathematics courses. A general variable skill test composed of 15 multiple choice verbal situations items was also administered. From the results of the study it was reported that both computer experience and mathematics experience were significantly correlated with general variable skills. Moreover, the relationship of computer programming experience to general variable skills was stronger than the relationship of mathematics experience to the variable skills.

Pea and Kurland (1984) analyzed the relationship between learning Logo programming and planning skills. Thirty-two students from a private school in Manhattan were selected for this study. The experimental group consisted of four boys and four girls from ages 8 to 9 years and four boys and four girls from ages 11 to 12 years. The control group consisted of four boys and four girls from each age group. The subjects in the experimental group received Logo instruction for two 45-minute sessions per week, while the control group received regular school instruction. Selection of experimental group subjects was based upon 1) the amount of time each subject had worked with Logo prior to the study and 2) teacher assessment of the reflectiveness and talkativeness of individual subjects. The students in the control group were selected based only upon the second factor.

A digit-span task and Block Design subtest were administered to measure subject cognitive styles (Pea & Kurland,

1984). The treatment consisted of two sessions in which the planning task was administered at the beginning of the first session and at the end of the second session. It was determined that the subjects who had spent a year learning programming skills did not significantly differ for various developmental comparisons of the effectiveness of their plans and their processes of planning than same-age students who had not learned computer programming.

Pea and Kurland (1984) also conducted a second study, which was similar to the experiment described above. The aim of the second study was to determine more closely the potential effect of programming skills upon planning skills. This study, within the same school and using the same teachers, took place one year following the first study. For this experiment, the teachers took more directive roles in guiding their students in learning Logo than in the previous study. Teachers gave weekly group lessons and demonstrated key computational concepts and techniques.

Pea and Kurland (1984) followed the pre-posttest design described for the previous study, with a total of 32 subjects participating in both sessions. Both the experimental group, which learned Logo programming, and the control group, which received regular schoolroom instruction, consisted of four boys and four girls from each of the age groups described above. To produce a more sensitive test for planning task analysis for the second session, an additional 32 students from the same classrooms were tested,

with four girls and four boys again selected for each group (i.e., experimental and control).

Two different versions of the chore-scheduling task were used for this second experiment. The first task was identical to the previous study task, whereas the second task was a computer-based chore-scheduling task that was designed to monitor and record subject performance. The original planning task was given to the experimental group at the beginning of the session. After six months, this subject group received Logo programming instruction. At the end of second session, all students were given the new planning task test. Statistical analysis failed to reveal significant differences between experimental and control groups. It was thus concluded that students who learned programming did not perform better in the development of planning skills than students who did not learn programming skills.

Summary

For the greater part, the studies reviewed did not indicate that learning computer programming skills had a measurable effect upon cognitive development. Note that several studies failed to provide information on the sample selection process or information on the socioeconomic status of the subjects. Moreover, some of the studies were based upon incorrect units of analysis, that is, some

studies, where a class was selected as the sample population, used the number of subjects as the unit of analysis. In addition, the reliability and the validity of some of the studies were not established.

Based upon a review of previously conducted research on the relationship between programming language instruction and problem solving skills, Palumbo (1990) stated that several issues were important to the successful design of programming language/problem solving research. These issues include methods of instruction, the length of the period of instruction, and the choice of programming language. In addition, an appropriate sample must be selected among groups of students who possess the requisite knowledge to acquire and learn a programming language.

For the most part, the research conducted in the studies reviewed was seemingly based upon the assumption that a relationship existed between learning programming skills and student cognitive development. Since Papert (1980) claimed that learning computer programming would accelerate student cognitive development, educators have sought to establish the validity of this claim. However, the studies which have been completed have not in each case collected the basic descriptive information required to provide full understanding of the subject behavior in programming. In point of fact, no completed research exists that has demonstrated precisely what students were learning while engaged in programming activities.

CHAPTER III

RESEARCH DESIGN AND METHODOLOGY

Introduction

The goal of this investigation was to provide an analysis of the thought processes of secondary level novice programmers while engaged in programming activities. Since the purpose of this goal was to generate hypotheses regarding the implications of these thought processes, the study was qualitative in nature. In the following sections, this chapter includes consideration of subject selection, problem consideration, methods, data collection procedures, and means of data analysis.

Subject Selection

For the sample population, a single class high school BASIC programming course in Oregon was selected. The 14 students enrolled in the class were from middle class families and ranged in grade levels from tenth to the twelfth grades. A first course in algebra was a prerequisite for enrollment in the BASIC programming course. The teacher had more than 10 years of experience teaching computer programming languages and computer application courses at the

high school level. Permission from the school district and from the parents of the subjects was obtained prior to the conduct of the study (Appendix A). A detailed description of individual subjects is included as Appendix C.

Data on subject thought processes while engaged in programming activities were collected twice during the first semester of the course: 1) during the first day of week 11 (hereafter referred to as Day I) and during the second day of week 16 (hereafter referred to as Day II). During both data collection days, the students worked in role-assigned partnerships wherein one student served as the problem solver and the second student served as recorder. These roles were switched following the solution of one problem. In this manner, each student was responsible for the solution of one problem. Since one student was absent the day the exercises were administered and only 13 subjects participated in the Day I problems, one student was required to work without a partner and attempted to solve both Day I problems (described in the following section).

For Day II, 14 subjects participated in the problem exercises. However, the data from only 13 subjects could be used. The data provided by one subject was excluded from the study since this subject was absent on Day I. In addition, a second subject had to leave the classroom after solving the first problem, thus leaving this subject's

partner to solve the second problem without assistance from a partner.

Problem Consideration

Four separate problems were considered by the subjects of this study. Based upon estimations of subject programming skill levels, the researcher and the teacher identified the problems, typical of those used for BASIC programming courses. Two problems were administered on Day I and a second two problems were administered on Day II. Each of the two problems used for data collection on each of the two days were different with respect to specifics, but were at the same time structurally similar. The problems are described in Table III-1.

Table III-1. BASIC Programming Problems.

Day	Session	Problem
I	I	KWH: Write a program that will compute an electric bill, giving the kilowatt-hour (KWH) usage (input by user). There is a basic charge of $3.00, plus usage billed at the following rates: 0-300 KWH @ $0.04237 per KWH 301-1,200 KWH @ $0.05241 per KWH over 1,200 KWH @ $0.06113 per KWH For example, a customer who has used 1,248 KWH will be charged the basic charge of $3.00, plus 300 KWH at $0.04237, 900 KWH at $0.05241, and 48 KWH at $0.06113, for a total of $65.81 (totals rounded to the nearest hundredth).

Table III-1 (continued).

Day	Session	Problem
I	II	MONEY: At the "$10 and Under" store, a customer buys an item that costs less than $10. Write a program that computes the amount of change a customer will receive from a $10 bill. The user should input the cost of one item in dollars (i.e., if the item costs 50 cents, the input is .50). The program should output the total amount of change and the quantity of each coin (excluding half-dollars) and $1 bills needed to make the correct change. For example, if the item cost $0.53, the customer should receive $9.47 in change, composed of nine $1 bills, one quarter, two dimes, and two pennies.
II	I	ELECTION: Write a computer program for the conduct of an election. There are three candidates: Milton P. Waxley (incumbent), Patricia Rhoder (progressive liberal), and Frederick "Red" Kemmeny (a reluctant candidate who filed at the last minute). The election is to be completed by votes determined randomly by the computer. Five hundred votes are cast? Who won the election? Give the user the option of repeating the election.
II	II	CARS: Write a program to help collect statistics on cars. Usually, the user sits at the terminal, looking through a window out into a busy street. Every time a car goes by, the user enters in a number indicating the manufacturer, either: 1) Ford, 2) General Motors, 3) Chrysler, 4) AMC, 5) German, 6) Japanese, 7) Other, or 8) Quitting time. After quitting time, the computer prints the number of cars that were seen in each of the first seven categories. From the previous problem, there is only one change: The user is the random number generator in the computer and the cars going by are randomly generated in the computer. In addition, the computer is willing to work until quitting time, which is also randomly generated. Randomly generate the number of cars going by. After quitting time, print the number of the cars that were "seen" in each of the first seven categories. Give the user the option to repeat the count of cars.

Content Validity

The content validity for the Day I problems given in Table III-1 was jointly established by 100 percent agreement among five computer teachers. The content validity for the Day II problems was jointly established by 100 percent agreement among the five computer teachers. The teachers thus agreed that these problems represented the content that they were designed to measure.

Equivalence

Equivalence for the Day I problems was jointly established by 80 percent agreement among five computer teachers, whereas equivalence for the Day II problems was jointly established by 100 percent agreement among five computer teachers.

Methods

Two data collections were used. Day I data were collected during the eleventh week of the fall semester, and Day II data were collected during the sixteenth week of the same semester. On each occasion, data were collected on Tuesday of that week, where the class schedule called for a double period (i.e., 90 minutes), thus facilitating subject participation in data collection on identical days.

According to the curriculum outline provided for the high school BASIC programming course, weeks one through 10 of the instruction were to emphasize command syntax and program structures in the BASIC language. During this 10-week period, emphasis was directed at writing simple programs which made use of the commands studied; no specific instruction in problem solving was given (see Table III-2). Following the initial data collection, five weeks of instruction were given in the techniques and strategies of structured programming as well as additional instruction with commands in BASIC language. Throughout both data collection periods, subjects were considered to be novice programmers. Thus, it was hypothesized that exploring subject thought processes at the end of the two periods of study would provide a clear picture of what the subjects had accomplished, thought or were thinking, or what they had learned by the end of the term. In addition, collection of the data provided information on changes in subject thought processes as they gained programming experience and problem solving strategy instruction.

To familiarize students with the data collection process for the experimental exercises, students practiced the protocol one day prior to the first data collection. At this time, demographic information on each student was collected, including mathematics levels achieved and prior computer experience in high school (Appendix B).

Table III-2. Classroom Schedule.

Week	Content
1-2	Introduction to computers: What is (are) a computer/computers? Organization History/uses
2-3	Input: BASIC, system, DOS commands PRINT/graphics, low-resolution
4-6	Output: Input: READ . . . DATA GOTO branching, unconditional
6	Test
6-8	Lab #1: Functions: INT, TAB
9-10	Branching: IF . . . THEN Trailer values STOP/TRACE
10	Data collection, Day I
11-12	Programming structures, looping: GOTO, counting/summing FOR . . . NEXT STEP GOSUB . . . RETURN Structured programming using subroutines Problem solving strategies
12-13	Programming structures, graphics High-resolution graphics/utilities Problem solving with graphics/movement Structured programming
14-15	Programming with functions ABS, SQR, SEG, RND Algorithm development: random number generation Problem solving strategies Structured programming
15	Data collection, Day II

Data Collection Procedures

For each data collection, a single day during one week, the class met for a double-period, doubling the length of instructional time available. During the first 15 minutes, the teacher read the instructions describing each subject's role in the data collection procedures. Subjects were then divided into groups of two by the instructor. One subject in each group was identified as the recorder and the second subject was identified as the problem solver.

The problem solver was asked to develop a computer program to solve a problem within a 30-minute period. The subject was allowed to solve the problem by any means, either working at the computer or working at a desk using pencil and paper. The problem solver's task was to solve the problem using a "think aloud" strategy, a method commonly used to investigate students' and teachers' thought processes. For the "think aloud" strategy, subjects were directed to speak out loud, describing their thoughts about the problem and methods of possible solution. The problem solver was asked to speak into a tape recorder during this process. For example, if the problem solver were thinking about input information required, then he/she would say:

```
To solve this problem, I will need the user's name.
So I will add these lines to the computer program
310 PRINT "Enter your name:";
320 INPUT NAME$
```

The recorder's task was to remind the problem solver to think aloud, speaking into an audio recorder while identifying thoughts and actions related to solving the problem. The recorder also took notes describing the problem solvers actions and made certain the audiotape was functioning correctly. The recorder was instructed not to assist the problem solver in solving the problem.

The recorder also was given the following instruction designed to help him/her obtain an accurate and complete recording:

1. What is the problem solver's student number?
2. What is your student number?
3. What time did the problem solver begin working on the problem?
4. Check: Is the tape recorder on "record" and is the problem solver speaking into the microphone?
5. Describe the problem solver's actions when first reading the problem:
   Did he/she write anything on paper?
   What did he/she write?
   Was the problem solver at his/her desk or somewhere else?
   Write down what the problem solver writes.
6. What time did the problem solver quit working on the problem?
7. Please retain copies of what the problem solver writes. Make certain that the problem solver

makes a printed copy of any program that was created. Include all the information in the packet when completed.

After the problem solver had solved the problem or had spent 30 minutes in attempting to solve the problem, the roles were reversed; the recorder during the previous session assumed the role of problem solver, while the problem solver from the previous session assumed the role of recorder. Again, the problem solver was given 30 minutes to solve the problem.

During the final 15 minutes, tape recorders were collected. Also, all materials, including problem solver hand written, printer output (i.e., program output, program source), audiotapes, and the recorder notes were placed in an envelope and collected by the researcher.

Means of Data Analysis

The audiotapes and the supporting information (e.g, the recorder notes and problem solver's program) were analyzed, using the qualitative technique described by Bogdan and Biklen (1982). Day I data were analyzed separately from Day II data. Following the separate analyses, patterns established within each period were compared to determine similarities and differences between subjects at both levels of novice programming performance. Thinking patterns identified in this situation were compared with

concepts taken from the literature of problem solving (Polya, 1988). From this comparative data, hypotheses were developed based on the patterns reflected within the data.

The analysis was initiated with the transcription of all of the audiotapes, each transcribed separately. Subjects were assigned a pseudonym corresponding to an identification number for purposes of identification. All of this information, including audiotape transcriptions, subject programs, and recorder notes, were placed in folders labeled with the name of each subject. The materials in each folder were duplicated. The original material was maintained in a safe place, whereas the duplicate information was used for purposes of data analysis.

After reviewing each of the transcripts accompanied by recorders' notes several times, a five-minute interval was identified as the smallest unit of time which could be used to usefully describe subject thought processes. Thus, each subjects' transcription with supporting information (e.g., recorder notes) were divided into five-minute intervals for purposes of data analysis.

Each five-minute interval for each subject was then compared to the parallel five-minute intervals for the other subjects, both singularly and collectively. The purpose of the comparisons was to search for similar behaviors among the students as they solved the problems. In addition, a profile for each subject was developed. The pro-

files contain demographic information on the subject and describe how the subject approached each of the problems.

The profiles were reviewed several times on several occasions in search of thought sequences commonly used among the subjects for the solution of the problems. Two sequences were thus created. Sequence one described how students approached the Day I problems and sequence two described how students approached Day II problems. The analysis of this material is presented in Chapter IV.

CHAPTER IV

RESULTS

This chapter reviews the results of the data analysis, provided in four sections. The first section provides a summary profile of each student's thought processes while solving each of the problems. The second section reviews the results that stem from the analysis of the Day I exercises, while the third section reviews the results derived from the analysis of the Day II exercises. A fourth section includes an analysis of the data with respect to the original questions.

Subject Thought Processing Profiles

A summary profile of subject thought processes includes a brief description of the approach each subject adopted for the problems considered. For purposes of identification and discussion, pseudonyms have been assigned to each of the subjects. Each subject solved at least one problem for Day I, November 14, 1991, and for Day II, January 19, 1992. On Day I, subjects worked on either the KWH or the MONEY problems, whereas on Day II subjects worked on the CARS or ELECTION problems (Table III-1). Profiles are

presented for, respectively, Day I and Day II. A detailed profile for each subject is provided in Appendix C.

AJ

AJ approached the MONEY problem by typing "10 HOME," followed by codes that were directly translated from the third line in the problem specification. Immediate execution of these codes resulted in a syntax error and halted the program. In actuality, AJ had more than one syntax error which required correction before receiving the correct response. As he reached a sentence in the problem specification that could not be translated directly into BASIC code, he guessed at code to be used, entering and executing each attempt. Typically, the result was transmitted with a syntax error. When the syntax errors were corrected, logical errors then became apparent. At the end of the period, AJ's program ran without syntax errors, but correct results were not produced.

On Day II, AJ attacked the CARS problem by immediately typing "10 HOME," followed by the BASIC coded algorithm to generate a random number, as described in the problem. When he executed these lines, he experienced logic errors. AJ's approach to the CARS problem was similar to that used for the MONEY problem. When he reached a sentence or phrase in the problem that did not resemble directly translatable BASIC codes, he provided a best-guess solution for that part. Logic errors were revealed when the program

output an incorrect result. These errors prompted AJ to ask for help from the teacher. Provided with assistance, AJ entered and executed changes in the codes, for the most part guessing at the proper changes rather than analyzing the process. At the end of this process, the program ran but did not produce correct results.

Ed

When Ed was given the MONEY problem, he immediately entered the codes that could be directly translated from the instruction in the problem specification. He executed this codes but did not mentioned whether he received syntax error or logic error. Ed ignored the problem since he was seemingly not convinced of the accuracy of the results; he continued to solve the problem. His strategy for solving the problem was to work with the problem line-by-line, sequentially entering appropriate codes. Once the codes for one line reflected no syntax errors, he would advance to the next line. Ed attacked the logic errors by typing and executing best-guess commands, but he continued to receive incorrect results. His next strategy was to ask the teacher for help, following which Ed continued efforts to solve the problem. Upon completion of the allotted time, Ed's program ran, producing the correct output.

Ed approached the CARS problem in the same way as the MONEY problem, immediately typing "10 HOME," followed by the codes that were directly translated from instructions

in the problem specification. Ed continued this strategy, sequentially analyzing problem instructions in order of appearance and then entering and executing the codes. Each time he executed the codes for one line, a syntax error was encountered. As for the MONEY problem, once the syntax errors were sequentially corrected, logical errors became apparent. Ed guessed at the commands, then entered and executed commands to debug the logical error. At end of the time, the program ran and was logically correct.

Frank

Frank attempted to solve the KWH problem by immediately typing the line "10 HOME." Next, he directly translated BASIC code for the line, indicating user input in kilowatt-hours. As Frank reached a line in the problem specification that could not be directly translated into BASIC code, he experienced problems creating codes for that line. Frank entered and executed codes which he guessed would solve the problem. Typically, this strategy led to the entry of syntax errors into the program. He sequentially corrected the syntax errors. Once the syntax errors were eliminated and the program was running, incorrect results were obtained. At this point, Frank recognized that he had logical errors in his program. His strategy was to seek the help of the teacher. With the teacher's assistance, Frank continued to work on the problem in a sequential manner, guessing at the changes to be made in each line of

the codes subject to correction. Once he had analyzed the codes for a single program line, he proceeded to the next line. Although Frank's program ran with no syntax errors, several logical errors prevented the output of correct results.

Frank approached the CARS problem by reading it and immediately asking the teacher for assistance in the determination of an algorithm to generate the appropriate random number. As for the KWH problem, Frank typed the first line "10 HOME," then entered an interpretation of the codes for generating a random number. This strategy was similar to the KWH problem solution, for which Frank attempted to create codes for problem lines that could not be translated directly from the problem specifications. As he entered and executed codes, syntax errors occurred; as for the previous problem, Frank sequentially guessed at corrections of the syntax errors while rerunning the program to check for results. He continued to experience logic problems while concentrating on one line of code at a time. Frank used a sequential strategy to solve the entire problem. At the end of the allotted time, his program produced the correct results, although it did not include an option to rerun the program.

Joe

When Joe received the MONEY problem, he immediately typed "10 HOME," followed by the codes that were directly translated from the third instruction in the problem specification. He sequentially entered and executed codes to solve the problem for each line in the problem specification. Once the codes worked properly, he advanced to the next line in the problem. Typically, Joe received syntax errors when the codes were executed. Correcting syntax errors then exposed errors in logic. At this point, Joe indicated that he could not find the problem and asked the teacher for help. With the teacher's assistance in identifying and correcting the logical error, he continued to work on the problem. In addition to asking the teacher for help, he guessed at logical error corrections, entering and checking the results by running the program. At the completion of the time period, the program ran and produced the correct results.

Joe approached the ELECTION problem similarly, typing "10 HOME" followed by codes that displayed a welcome message on the screen. He entered codes as directly translated from the instructions in the problem specifications. As the codes were executed, syntax errors occurred. After sequentially guessing at a correction for the syntax errors, the program was executed and the correct output resulted. As for the MONEY problem, Joe encountered a

sentence in the problem that could not be directly translated into BASIC codes and had to ask the teacher for assistance. This occurred in the instance of each logical error. Guessing at appropriate changes to the program frequently resulted in syntax errors. At the end of the period, his program did not run because several syntax errors remained.

Jon

Jon began to solve the KWH problem by reading the problem several times, then entering codes to display a welcoming message on the screen. Next, he entered codes directly translating the first instruction in the problem specifications. Once he reached a sentence in the problem specification that could not be directly translated into BASIC codes, he guessed at the correct codes to solve the problem; executing these guesses resulted in syntax errors. Jon sequentially guessed at the corrections for syntax errors and executed the program, only to discover the results were incorrect. Again, he guessed at codes required to solve logic errors. As a result of logic problems, the program did not produce correct results within the allotted time.

Jon approached the MONEY problem in the same fashion, typing "10 HOME," followed by codes to display a message on the screen. He entered the codes as directly translated from the third instruction of the problem specifications.

As for the previous problem, he reached a part in the problem that he could not directly translate into BASIC codes and guessed at correct solutions. Execution of these codes exposed logic errors. He then guessed at codes to fix the logical errors, but was still not able to obtain the correct results. At this point, he had no further strategies for obtaining a problem solution. Although his program ran without syntax errors, the correct results were not obtained.

Jon began the ELECTION problem by typing "10 HOME," followed by a coded algorithm which translated the first instruction in the problem specifications to randomly generated numbers in the range of 1 to 3. Execution of the code was stopped by syntax errors. Once the syntax errors were removed, the results were correct. He proceeded to solve the next part of the problem, but was blocked because the specification for this part was not straightforward with respect to the translation of the BASIC code. Thus, he adopted the same strategy as for the MONEY problem, guessing and entering commands, the execution of which was terminated by syntax errors. Once the syntax errors were corrected, logic errors were revealed. Again, Jon guessed at correct solutions. The program ran, but failed to produce the correct results.

Ken

Ken approached the MONEY problem by entering codes that were translated directly from the third instruction in the problem specifications. He needed to correct several syntax errors before discovering an error in his logic. He could not correct the logical errors, but continued to solve the problem. As he reached the lines in the problem specifications that could not be directly translated into BASIC codes, he entered and executed best-guess codes, but received incorrect results. At this time, he typed any codes that he could think of, but his program was not able to produce the correct results. Ken asked the teacher for help. Following assistance, Ken continued to solve the problem by examining each sentence in order of appearance within the problem, then entering the codes for each sentence in sequential order. His program then ran and generated the correct results.

Ken solved the CARS problem in a similar fashion, entering codes that could be translated directly from the problem specifications. Once more, as he reached a part of the problem that could not be coded in a straightforward manner from the problem specifications, he guessed at the codes, entered and executed the codes, and then typically obtained incorrect results. At this time, he had to ask the teacher for help in correcting the logic errors, following which corrections were obtained. However, at the

end of period he was not able to provide the option to allow the user to rerun the program.

Lee

After reading the KWH problem, Lee immediately requested the teacher's assistance to understand the problem. Following this discussion, he typed "10 HOME" and the codes for inputing information, as directed in the problem specifications. As soon as he reached a line that could not be translated directly into BASIC code, he apparently had no idea how the problem could be solved. His strategy was to seek help from the teacher. With the teacher's help, Lee continued to work on the line in question, but then experienced syntax errors. Once the syntax errors were sequentially corrected by guessing at correct codes, logical errors became apparent. Again, Lee asked the teacher for help. He continued to work on the problem line-by-line, attempting to create BASIC codes to solve the problem. At the end of the period, his program ran with a logically correct solution.

As for the KWH problem, Lee began to solve the ELECTION problem by typing "10 HOME," followed by the code directing the instructions in the problem specifications. As soon as he encountered a part of the problem specification that could not be translated in a straightforwarded manner, he asked the teacher for assistance. He realized that he had logic errors in his program and that assistance

would be required. With this help, he was able to return to a sequential analysis of each problem specification. At the end of the period, his program produced the correct output.

Mark

Mark approached the KWH by typing "10 HOME," followed by codes directly translated from the first instruction in the problem specifications. He worked sequentially with each instruction, attempting direct translations into known BASIC codes. He guessed at codes, which resulted in syntax errors upon execution. He sequentially guessed at corrections for the syntax errors only to encounter logic errors. At this point, Mark asked the teacher for assistance. With this help, he was able to continue the problem solution by analyzing each line in the problem in order of appearance in the problem specification and then entering an appropriate code. At the end of the period, his program was free of syntax errors, but failed to generate the correct output.

Mark began solving the ELECTION problem in a similar manner, typing "10 HOME" followed by instructions translated from the problem specifications. He used the same strategy as for the KWH problem, sequentially analyzing each sentence in the problem specification and then entering codes for that sentence. Once the code had produced a correct output, he proceeded to the next part of the problem.

He then sequentially guessed at corrections for the syntax errors and executed the program until the correct results were obtained. At the end of the period, the program ran in a logically correct fashion; but he did not include the option to allow the user to repeat the election process.

Rick

Rick worked on the KWH problem by typing "10 HOME," followed by codes translated directly from the first instruction of the problem specifications. Once Rick reached a part of the problem specification that could not be translated in a straightforwarded manner, he guessed at correct solutions; execution was terminated by syntax errors. After correcting the syntax errors, he encountered logic errors. Again, he guessed at appropriate codes to debug the logic problem and his program continued to produce incorrect results. At this point Rick asked the teacher for assistance. With this help, he was able to correct the logical errors and by the end of the period a logically correct problem was obtained.

Rick approached the counting CARS problem by typing "10 HOME," followed by codes translated from the instruction in the problem specifications. His approach was similar to that used for the KWH problem. He continued to enter and execute commands and to encounter syntax errors. Correcting the syntax errors revealed logic errors. By the end of the time period, he was able to solve the problem

with a logically correct program, but one which did not include the option to allow the user to repeat the counting process.

Sam

Sam confronted the MONEY problem by typing "10 HOME," followed by codes translated directly from the third instruction in the problem specification. Each time he reached a part of the problem that could not be solved by direct translation, errors resulted. He guessed at the codes, entered and executed them and continued to experience errors. Sam's strategy was to ask the teacher for help. With this help, Sam continued to solve the problem. Although his program ran without syntax errors, it was incomplete.

Sam approached the ELECTION problem by typing "10 HOME," then asking the teacher for help in attacking the problem. This approach was similar to his strategy with the MONEY problem. He became sidetracked with entering codes to display user friendly messages on the screen. Execution of the codes resulted in syntax errors. He sequentially attacked the syntax errors by guessing at correct commands, but was unable to create a logically correct solution. He then asked for help in correcting the logic errors. His attempts at solutions then resulted in new syntax errors. At the end of the period, his program was halted by a syntax error. In actuality, for his final

solution he had several syntax errors as well as a logic error.

Sue

Sue attempted to solve the KWH problem by typing "10 HOME," followed by codes translated from the problem specifications. Sue was stopped by an instruction that required analysis before it could be translated into BASIC codes. Her strategy was to ask for help from the teacher. The teacher helped her to correct the BASIC code for that line. She entered and executed the codes, then encountered syntax errors. Sue asked her teacher for help in correcting more than one syntax error. Once the syntax errors were corrected, the program did not run logically. Sue then asked her recorder to help her correct the problem. Her strategy to solve the problem was to ask either the teacher or her recorder for assistance and then to enter specific BASIC codes. At the end of the period, several syntax errors prevented the program from running.

Sue approached the CARS problem similarly, typing "10 HOME" as the first line. Next, she asked her recorder about the format of the algorithm to generate a random number. Whenever she encountered parts of the problem that required analysis, her confusion was apparent. As before, her strategy was to seek help from the teacher. With this help, she guessed at new code combinations and then experienced syntax errors as lines were entered and executed.

Sue sequentially guessed at corrections for the syntax errors, but still received an incorrect output. She was unable to determine her errors in logic and again asked the teacher for assistance. At the end of the period, the program was free of syntax and logic errors.

Tom

Tom also began solving the KWH problem by entering "10 HOME" and then translating the first instruction in the problem specifications. As he reached a part in the problem specification where this strategy failed, he guessed at codes. He wrote the entire program on paper, then entered and executed the program at the computer. The run was halted by a syntax error. With the teacher's help, he was able to correct his syntax errors, only to encounter logic errors. The program did not output the correct result. Tom again asked for help and was able to correct the errors in logic. He continued to sequentially enter codes to execute checks. He guessed at correction to both the syntax and logical errors. As a result of several logical errors, at the end of the period his program ran, but did not provide the correct output.

Tom approached the ELECTION problem using a similar strategy, this time working at the computer rather than initially writing out the codes on paper. He entered the codes translated sequentially from parts of the problem specifications. As in the KWH problem, when he reached a

part of the problem specification that could not be solved in this manner, he guessed at correct codes and experienced syntax errors. Once the syntax errors were sequentially corrected by guessing at appropriate commands, his logic errors were revealed. He guessed at commands to overcome the logical problems. Although his program ran, it did not produce the correct output.

Tony

Tony initiated his solution to the MONEY problem by typing "10 HOME," followed by codes directly translated from the instructions in the problem specifications. When he experienced logic errors, he asked the teacher for assistance. Tony was then able to fix the logic errors. Each time he reached a sentence in the problem specification that could not be directly translated into BASIC codes, he guessed at sequences of commands and then experienced syntax errors. Once the syntax errors were eliminated, Tony encountered logical errors. He guessed at command solutions, entered executable codes to overcome the logical problems, and ultimately achieved a logically correct program.

Tony approached the ELECTION problem similarly, entering "10 HOME" followed by codes translated directly from the problem specifications. He executed the codes but experienced syntax errors. Tony sequentially corrected the syntax errors until a correct result appeared. As for the

MONEY problem, Tony attacked the problem line-by-line, guessing at correct solutions, then entering and executing lines of codes. Once the codes produced the needed output for that part of the problem, he advanced to the next part. Typically, he entered syntax errors. Once these errors were corrected by guessing at and entering appropriate codes, his logic errors were revealed. Tony continued to guess at commands to debug the logical errors, and by the end of the period his program produced the correct output.

Analysis of Day I Results

The analysis of Day I results for all students revealed three specific strategies displayed by the majority of the students when solving the problems. These strategies are presented in the following sections, accompanied by supporting evidence. Following the description of each strategy, the typical programming sequence followed by the subjects in solving the problem is also described.

Coded Thinking Strategy

Students approached the Day I problems thinking in terms of BASIC code language, rather than creating a plan for solving the problem or for the development of a solution algorithm. Typically, the subjects entered codes as if they were directly translating the words of the problem

into BASIC code. All students began the problem by typing "10 HOME" as the first line.

Situation 1 (Mark, KWH)

Mark said: I am going to start out by asking how many kilowatt-hours are used. I am going to say INPUT X.

Situation 2 (Ken, MONEY)

Ken said: 10 HOME, I need to type NEW. Then HOME (subject silent for 30 seconds). I am asking for INPUT the item cost.

Situation 3 (Rick, KWH)

Rick said: I am typing INPUT statement to ask the user to enter the number of kilowatt-hours using K as the variable.

Debugging Strategies

Subjects typically developed two strategies for attacking errors in the program, the guess-and-check and teacher assistance. The guess-and-check strategy was applied to both syntax and logic errors where subjects made a "trial guess" and then ran the program to check to see if the error was corrected. Basically, they guessed, entered, and then executed different codes each time, until accurate codes were found. Generally, they tried to debug the program by changing variables or by adding different instructions to determine whether the program would generate the correct results. When subjects encountered syntax

errors, they sequentially guessed at corrections. Once the syntax errors were corrected, logic errors became visible. Subjects were able to discover logic errors by the output of incorrect results; however, the location of the error was not pointed out by the computer. At this point, subjects were frequently stumped and asked the teacher for assistance. The subjects' principal strategy to overcome logical errors was to seek help from the teacher or to use the guess-and-check debugging strategy.

When syntax errors occurred, guess commands were entered and the program was executed to determined the error had been fixed and that the program would output a correct result. The subjects repeated the guess-and-check strategy as necessary. Subjects continued with this strategy until a proper sequence of commands was identified. When the subjects found this strategy to be ineffective, the next step was to seek help from the teacher.

Syntax Error Strategy

Subject thought processes were continually interrupted by syntax errors, which involved errors such as placement of colon or parenthesis or misplaced commands. Correcting these syntax errors seemed to be the first challenge most of the subjects faced in solving the problem. They sequentially corrected these types of errors by using a guess-and-check strategy, allowing the computer to point out errors. The BASIC interpreter in the computer halts

program execution at the point of the error. For both problems, students had problems understanding the grammatical and structured rules using the integer (INT) function; in fact, the most common syntax errors were related to the INT function. Subject strategies were to experiment with this function by guessing at correct codes or to ask the teacher for assistance in correcting the errors.

Situation 1 (AJ, MONEY)

Recorder wrote:

Syntax in 35

Changing same line to LET statement

AJ's program:

Code before correction: 35 10 - B =

Code after correction: 35 LET B = 10

Situation 2 (Ken, MONEY)

Observer comments: The student was having difficulty applying the INT function in the program to compute the number of coins in the customer's change. He was trying to figure it by experimentation.

Ken said: Oops! I am experimenting in rounding to the hundreds places. Oh, man (subject quiet for two minutes). I know what I did wrong. I think I fixed it. Let's see here. Oops! Syntax error of some sort. I think I fixed it. Let's see. I got it. Now I got to print first. Now I gotta

figure out this one dollar bill thing. I hope that it is easier than the amount of change. Go ahead and list. I am just playing around with something. That thing, Oops! integer (subject was quiet for 30 seconds). Oops, I forgot one thing. I got to print it. Integer of C. Oops!, I am trying to screw with the integer but I am not going to get it. I got to get it by myself. This is not gonna work. I already tried.

Recorder notes:

Experimented with rounding. He forgot how though.

Experimented with integer

Typed in Q = INT (C - P)

Trying to print number of quarters, PRINT Q.

Situation 3 (Tom, KWH)

Observer comment: The student used a string variable for numeric value. With the teacher's assistance, he changed all the string variables to numeric variables.

Tom said: I do not know why it does the syntax error on line 30. A string can not express.

Teacher said: You do not need the dollar signs.

Tom said: I am taking out all the dollar signs out of all my variables. I do not need the dollar signs for number.

Code before:

```
20 INPUT "NUMBER OF KWH
USED:" A$
25 R = (.04 + 3 ) * A$
30 IF A$ < = 300 THEN  PRINT R
40 P = (.05 + 3) * A$
50 IF A$ > = 301 AND A$ <= 1200 THEN
     PRINT P
```

Code after:

```
20 INPUT "NUMBER OF KWH USED:" A
25 R = (.04 + 3 ) * A
30 IF A  < = 300 THEN  PRINT R
40 P = (.05 + 3) * A
50 IF A > = 301 AND A <= 1200 THEN
     PRINT P
```

Situation 4 (Mark, KWH)

Observer comments: Mark had syntax errors as a result of not including the colon to separate commands. Also, he needed to add the basic charge to the total charge.

Mark said: Right now I am going to test it one more time. Something is incorrect. I think it's because I do not have parenthesis . . . . So I am back to edit 70. I just started to do this. I need to put parenthesis around. I got to test again. Still have syntax error (subject was quiet for one minute). I am gonna go back and change 50. So if X is greater than . . . .

Recorder notes:

He is pondering

Syntax error in 70

Thought it was because he did not put colon between total and GOTO 100

Did same to line 80 and 90"

Code after correction:

```
70 T = (X * 0.04237) + 3: GOTO 100
80 T = (X * 0.052441) + 3: GOTO 100
90 T = (X * 0.06113) + 3: GOTO 100
```

Logical Error Strategy

Subject thought processes were interrupted when their programs did not output correct results. Unlike the identification of syntax errors, difficulty was experienced determining the location of the errors since the only way the students recognized the logic errors was by the output of incorrect results. The computer does not identify the location of this type of error. Typically, subjects attacked this type of problem by seeking help from the teacher. A second approach was to apply the guess-and-check strategy, the use of which often resulted in adding syntax errors to the logic error. Subjects had problems breaking the number of kilowatt-hours into different segments and then computing the charge for the number of the kilowatt-hours in each segment. A common problem was that they failed to add the basic charge to the total charge. In the

MONEY problem, subjects had problems computing the number of coins in the customer's change.

Situation 1 (Joe, MONEY)

Observer comment: The teacher explained to the student how to compute the change. The cost of the item was $7.45, so the change was calculated by subtracting $7.45 from $10, which was equal to $2.55.

Joe said: I need to find out how many coins she will or the person will receive like dollars or coins. (subject was quiet for one minute). This is the tricky part. I am still trying to figure out how to do this (subject was quiet for one minute).

Teacher: [Teacher is helping ...]

Joe said: I know how much change I have but I am having problems. (Subject was quiet for 10 seconds.) I am having trouble, like finding out how to set up how many bills?

Teacher said: It's right there, there will be 255 cents for the change. $10 - $7.45 for change. Integer will take care of the dollar amount.

Joe said: So I should print this and . . . B= INT (C) times 100.

Situation 2 (Lee, KWH)

Observer comments: Lee had a problem rounding off total charges to two decimal places and had to ask the teacher for help. After teacher assist-

ance, Lee edited lines 610, 620 and 630 to add an integer command. He then executed the program but received incorrect results since he did not add the basic charge ($3.00) to the total charge.

Lee said: I am trying to print the total charge G dollars from these lines 610, 620, and 630. I do not know how to do that. Teacher, can you tell me the command that round off the total charge?

Teacher: I can tell you that, remember the integer command?

Lee: Oh.

Teacher: You know how to do that with the integer command. Plug it in front of your total charge.

Lee: I need to edit lines 610, 620, and 630 to add the integer commands. I am going to run the program. I gotta to reenter some values for the kilowatt-hour charge. My program is not producing a correct result. (Lee was quiet for two minutes.) I think I need to add the basic charge to fix the problem.

Situation 3 (Rick, KWH)

Observer comments: In this situation the subject had two logic problems. First, he did not add the basic charge ($3.00) to total charge. The second problem was in rounding the total charge to two decimal points.

Rick: I need to add $3.00 charge. I need to edit line 50 so I can include the $3.00 charge. Run the program. It is not working, but I need to round off the number. B equal integer of B times 100 and then divide by 100 plus 100. Still not working (teacher helping ...). I am running the program again and see it should round off to two decimal points (Rick was quiet for two minutes). It is not working. I need to edit line 55. B equals integer of B ... I got to include the basic charge ($3.00). B times 100 and divided by 100, okay I got it. Run the program. It is not working. Editing 60 to round off the decimal point. Run the program. Still it is not working.

Situation 4 (Ed, MONEY)

Ed said: I have got the dollars now and it works. So now I got to figure out the quarters. I think I will ask the teacher.

Teacher said: You need to get the decimal part from the change, I know it is hard, but you have to think about it.

Ed said: I am working on the quarters (subject typing but not talking for one minute). I am trying to figure out the quarters. I am separating the new change. The number of quarters is the decimal part in the change.

Programming Process

When subjects were presented with the problem, they typically moved directly to the computer before reading. After an initial reading, they appeared to immediately accept the problem by entering "10 HOME" into the computer to begin a solution. They approached the problem thinking in codes rather than creating a plan or developing an algorithm to be followed for solution of the problem. Their thoughts were typically coded in the BASIC language rather than in the English language. Furthermore, students entered codes that easily translated the instructions in problem specifications into the BASIC code. For example, for the KWH problem, the user was asked to input the kilowatt-hour usage. This instruction was translated as INPUT K. In the MONEY problem, the customer was asked to input the cost of the item in dollars. The BASIC code for this type of action is INPUT COST. At this point, the problem seemed straightforward since the subjects had not thought the entire problem through clearly. In fact, they thought only about the codes that could be translated from the first instruction in the problem specifications for the problem.

When the subjects reached a part of the problem specification that was not clearly translatable into BASIC code, they encountered solution problems. They resorted to guessing and entering codes. For the most part, several syntax

errors were identified when these codes were executed. At this point the subjects did not have a clear understanding of the syntax of the BASIC language commands. Therefore, they continued to make syntax errors. The subjects struggled with correcting these syntax errors, using the guess-and-check strategy. The teacher was often asked for assistance in correcting the syntax errors.

After the syntax errors were sequentially corrected, logic errors became apparent from the output of incorrect results. Students again used a guess-and-check strategy to debug these errors. When this technique did not prove successful, subjects were not able to correct their results on their own. Their inability to solve logic errors by this method frustrated the subjects. As a result, they resorted to asking the teacher to explain concepts. For example, despite the fact that each problem contained written examples, the teacher was typically asked to explain the conversion of kilowatt-hour usage to a total charge, or the conversion of the customer's change to a number of coins. Teacher assistance seemed to encourage the subjects to continue working with the problem.

In general, the subjects perceived the problem as a sequence of several parts, rather than as an integral whole. They read the specifications for each part or line or sentence in the problem and then attempted to create BASIC codes to solve the individual parts. Upon successful completion of one part, they advanced to the next part of

the problem. Subjects used this strategy until the entire problem was solved.

Analysis of Day II Results

The analysis of Day II results for all subjects revealed three specific strategies that the majority of the subjects displayed when solving the problems. These strategies, with supporting examples, are presented in this section. The general programming sequence observed by the subjects for solving the problem follows the description of the characteristics.

Coded Thinking Strategy

For the Day II exercises, the subjects continued to approach the problems with coded thought, rather than creating a plan or developing an algorithm to solve the problem. They began the problem by entering code to translate instructions for the problems.

Situation 1 (Jon, ElECTION)

Jon said: So I 'll go with 10 HOME. Now there are three candidates. So 500 votes will be cast.

Situation 2 (Lee, ELECTION)

Lee said: I guess I will start at "10 HOME" then I will just write the names of the candidates. I need to input the codes for the random generator algorithm commands.

Situation 3 (Rick, CARS)

Recorder wrote:

Set up random number statement and thought about the problem.

He then set up some values for X and other variables.

Rick's program:

```
10 HOME
20 PRINT TAB (20) "CARS STATISTICS"
30 X = INT (RND (69) * B ) + 1
```

Debugging Strategies

Debugging strategies were used to correct both syntax and logical errors. Basically, subjects guessed at changes, then entered and executed different codes until the proper commands were identified. In addition, to follow the guessing strategy, subjects changed variables or added different instructions and checked whether the program would generate desired outcomes.

Syntax Error Strategies

For Day II, syntax errors were not considered a problem for the subjects. When compared to Day I, during which the subjects struggled with syntax errors, they seemed to be more familiar with the syntax of the BASIC commands and experienced fewer syntax errors. For Day II, the subjects

were able to correct syntax errors without seeking assistance from the teacher.

Situation 1 (AJ, CARS)

Observer comments: During Day I, AJ experienced a syntax error writing the codes that assigned a value to a variable. During Day II, AJ did not experience problems using the BASIC commands in his program.

Situation 2 (Ken, CARS)

Observer comments: During Day I, Ken experienced difficulties applying the program INT function to compute the number of coins in the customer's change. He tried to figure it out by experimentation. During Day II, Ken's thought processes did not demonstrate any syntax language difficulties.

Situation 3 (Tom, ELECTION)

Observer comments: Tom used a string variable for the numeric value. With the teacher's assistance, he changed all string variables to numeric variables. During Day II, Tom's thought processes did not demonstrate language syntax problems.

Logical Error Strategies

These types of errors stumped the subjects. Requesting help from the teacher was again a typical response. The teacher identified points to consider in correcting the

errors. With this assistance, subjects returned to the guess-and-check approach.

Subject thought processes were interrupted by logical difficulties. Indeed, subjects did not recognize logical difficulties or even believe they had logical problems until all of the syntax errors were corrected. Once the syntax errors were corrected, their programs did not produce the correct results and the subjects had no concept regarding the source of errors since the computer did not identify the location of the error or errors. Frequently, the subjects' first strategy was to seek help from the teacher. The teacher not only identified the likely sources of error, but also re-explained the problem. With this help, subjects continued to solve the problem using the guess-and-check method as a debugging strategy.

Situation 1 (Ken, CARS)

Observer comment: Ken tried to check the brand name of each car randomly. Once he identified the kind of car, he added them together. He used the same variable name for each counter in the program, which generated logic error since each counter had the same value for the number of cars.

Ken said: I am just gonna do each of these using the first letter of each one (each make of car). Ok! I just go back and recharge line-by-line. Oh! This is like the same thing I am doing to each

car. . . have a random number for each (for each car type). Oh! not random, but check each car to see if it's non-biased. Am I using my first two letters of each car's part?

Ken's program:

Before corrections:

```
60 IF CO = 1 THEN GM = F + 1
62 OF CO = 2 THEN GM = GM + 1
64 IF CO = 3 THEN GM = CH + 1
```

After corrections:

```
60 IF CO = 1 THEN F = F + 1
62 OF CO = 2 THEN GM = GM + 1
64 IF CO = 3 THEN CH = CH + 1
```

Situation 2 (Joe, ELECTION)

Joe said: Now I got to find out who wins the election. Teacher, um. Ok! I have my random numbers generated 500 times, but now I got to figure out how to determine the winner.

Teacher said: You have to check the random number and if the number is equal to one, then the count goes to the first candidate and if the random number is equal to two, then the count goes to the second candidate and so on.

Joe said: I am getting on track. Ok! 80, if X = 1 then A equal A plus one. I do not know what to write. I do not know what I am doing. Teacher, I got my random number from 1 to 3, but I am

trying to figure out how to add them for each candidate.

Teacher said: If the random number is equal to two, then you add one to the second candidate's counter. And if the random number is equal to three, then you add one to the third candidate's counter.

Situation 3 (AJ, CARS)

Observer comments: AJ was using a loop that instructs the computer to generate a random number 100 times rather than checking the time to quit counting cars. The teacher advised him to check if the random number was equal to 8, when it would be time to quit counting the cars. After the teacher helped AJ add line 57, he checked the random number.

AJ said: I do not know when to stop counting cars. I do not have a variable that does this.

Teacher said: What you need is to check if the random number is equal to 8 and if it is, then you need to quit counting cars. Otherwise you need to repeat the procedure of generating a random number and identifying the type of car.

AJ said: All that I need now is to check if the random number is equal to 8 in order to quit counting the cars.

Teacher said: You do not need the loop in the program. You need to ask if the random number is equal to 8 then go to END.

Programming Process

Similar to Day I, when subjects were presented with the problem, they moved directly to the computer before completely reading the problem. Once they had read the problem, they seemed to immediately accept the problem by entering "10 HOME" into the computer to begin a solution. They worked with the problem thinking in BASIC code, rather than in the English language. Moreover, they clearly translated the problem specifications into BASIC codes, just as was the case for Day I. For both the CARS and ElECTIONS problems, they needed to create a random number. Students translated this instruction into the random number algorithm coded in BASIC. They were not concerned with the entire problem, but only with the part for generating a random number.

As the students reached a part, or a line or sentence in the problem, that was not clearly a translation of BASIC code, they guessed at appropriate codes. Upon execution, these codes were identified as containing syntax errors. For Day II, subjects did not experience problems, nor did they ask the teacher for help correcting syntax errors. They were clearly able to deal with syntax errors, includ-

ing checking and correction. However, just as for Day I, the elimination of syntax errors identified logic errors.

Subjects continued to be stumped by logic errors. Their first strategy was to seek help from the teacher. The teacher helped them to see the error and then provided an explanation for correction. Students then returned to a the guess-and-check strategy to correct the problem. Often, this strategy was not successful, and the students were confused. They again requested help from the teacher. The random number algorithm in each problem was particularly difficult for the subjects. The teacher's assistance encouraged students to continue working with the problem.

In general, the subjects perceived the problems as composed of several parts rather than as an integral whole. They read sentences or instructions in order of appearance in the problem description. They attempted to create BASIC codes to solve each part of the problem sequentially. Upon successful completion of each part, the subjects advanced to the next part of the problem, continuing the same strategies until the entire problem was solved.

Data Analysis for the General Questions

Several general questions were posed for this investigation:

1) Do students' thought processes change as a result of additional instruction and experience?
2) What are students thinking while actively engaged in program development?
3) Do students' thought processes resemble the thought processes involved in generic problem solving (i.e., planning procedures) while they are actively engaged in program development?
4) Are students' thoughts a reflection of scientific programming codes (i.e., language syntax) or of the problem that is posed?

For the sample considered in this investigation, subject thought processes remained unchanged, with the exception of those strategies developed to solve syntax errors. For Day I, subjects struggled with the correction of syntax errors, often using the guess-and-check strategy or asking the teacher for help. For Day II, the subjects appeared to be more familiar with the BASIC commands and thus corrected the syntax errors with greater confidence and efficiency.

The thinking processes used to solve the problems were not equivalent to the problem solving processes identified by Polya (1988). Rather than devise a plan for solving the problem (i.e., as Polya had indicated should be done), the

subjects first concentrated upon the instructions, thinking in BASIC codes. They did not derive a plan in English, then carry it out by converting it to BASIC codes, which would have been the likely Polya problem solving solution. Furthermore, the subjects did not review their experience with the determination of a solution after completing the exercises. Once a program solution was obtained that returned a correct result for the situation, they concluded that this solution would work for all cases.

For this group of subjects, their thought processes were clearly guided by the BASIC code commands with which they were familiar. In other words, they applied what they knew about specific commands to create a solution, focusing upon the syntax of the BASIC codes rather than upon the problem. Sequentially, they were influenced in this process by the instructions in the problem specifications. They considered the first instruction to be the first part of the problem. After translating that part to BASIC, they then moved to consideration of the next sentence of the problem statement.

CHAPTER V

DISCUSSION AND CONCLUSIONS

The purpose of the present study was to investigate student thought processes while engaged in computer programming in order to generate hypotheses for future research on the relationship between computer programming and problem solving. This chapter is organized in four sections. The first presents a discussion of strategies and programming processes identified in this study. The second considers the limitations of this study. Recommendations for future research are presented in a third section, and the fourth section considers the implications for science teaching and science educational research drawn from the results of this study.

Strategies and Programming Process

Analysis of novice programmers' thought processes in the conduct of the present study revealed specific strategies and processes utilized in solving problems. Subjects thought about the problems in terms of BASIC code and exhibited specific debugging strategies when dealing with syntax or logic errors within the program. Moreover, students

exhibited specific processes or patterns when solving the programming problems.

Coded Thinking Strategy

The novice programmers who constituted the subjects for this study immediately accepted the problems presented them, thinking about them in terms of BASIC code. As they read lines in the problems and discussed them for the tape recorder, they spoke in BASIC codes rather than in complete sentences. This predisposition to coded thinking was obvious since the students spent most of their time on the computer entering codes about which they were relatively unsure. As they reached parts of the problems that could not be readily translated into BASIC code, difficulties were experienced with the problems. Other investigations have suggested that novice programmers lack the tools necessary to construct intermediate states between problem specifications and program codes (Dalby, Tournaire, & Linn 1980).

The coded thinking strategy used by the novice programmers may be similar to strategies used by students when learning a foreign language. When students translate from their native language into a foreign language, they not only must learn the foreign word for the translation, but must also learn the rules and grammar of the foreign language in order to complete their translation. As Oliva (1969) noted, to conduct effective translations, students

must be familiar with vocabulary and grammatical constructions as well as work with material that is within their ability level. In the translation of text from one language into another, students must first comprehend the source of the text, and then proceed with the translation paragraph by paragraph, establishing a list of unknown words and expressions in the order in which they are encountered in the text (Cordero, 1982).

Novice programmers approach the solution to a programming problem by attempting to translate the problem statement from their native language into the programming language. They must learn the actual BASIC commands as well as the grammar (or syntax) of the language to translate the native language statement contained within the problem. Specifically, when solving problems for the first data collection point, the subjects were aware of the BASIC commands, but they did not have a good understanding of the syntax required for the use of those commands. Rather, they used the coded thinking strategy of translating word-for-word from the problem statement to identify program statements for the solution of particular parts of the problem. As a result, a large number of syntax errors were committed as the codes were entered and as the words from the problem statement were translated into BASIC statements. The number of syntax errors that were observed supported the notion of students using a coded thinking strategy.

The question of what levels of thinking were exhibited by students using this coded thinking strategy remains to be considered. In sequence order, from the most sophisticated type of learning to the least, Gagné (1970) posited eight levels of learning: problem solving, principle or rule learning, concept learning, discrimination learning, verbal association, chaining, stimulus-response, and signal learning. According to this model, learning cognitive strategies is linked to rule learning or to problem solving learning.

In problem solving, the most sophisticated type of learning, the learner applies principles or rules to solve problems (Gagné, 1970). However, before the learner can solve the problem, the learner must understand the rules. Thus, learning the rules or principles is the second most sophisticated type of learning. Principle or rule learning consists of relating two or more concepts in the awareness that a concept must be learned and understood before it can be related. Hence, concept learning is identified as the third most sophisticated form of learning. Any number of times, the learner is required to identify responses to a number of different stimuli that may resemble each other to a greater or lesser degree. Thus, discrimination learning is the fourth most sophisticated learning. Then, any number of different stimuli need to be associated or linked to solve a problem. Ergo, verbal association learning becomes the fifth most sophisticated form of learning. More-

over, stimulus-response factors must be connected or chained together. Consequently, chained learning is the sixth most sophisticated learning. The learner must acquire a precise response to a discriminated stimulus. Therefore, stimulus response is the seventh most sophisticated type of learning. Finally, the learner has to make a general and diffuse response to signals, placing signal learning as the eighth most sophisticated form of learning.

The coded thinking strategy is similar to activities involved in Gagné's (1970) verbal association learning. Within this model, verbal association learning is marked by three-link chaining as follows: A previously discriminated verbal element function provides the initial stimulus; previously learned connections act as a second element of the chain; and a mediating (or coding) connection, supplied either by the learner or the instructor, links the two together. To learn programming, the problem statement provides the verbal element for the first link; the second link is then provided by previous instruction and practice creating statements using BASIC code to translate verbal elements; and the connecting link is the actual BASIC code, elicited by the verbal element in the problem situation. A problem statement, such as "input the student's name," provides the first link; previous practice with statements to input information using BASIC code is the second link; and the BASIC code INPUT connects the first and second link. As a result, the subjects voiced a mixture of the English

statements and coded translations when speaking into the recorder. This verbal association, or coded thinking, was continually used by the novice programmers in this study.

From Gagné's (1970) description of the hierarchy of learning, in contrast to the high level strategy required at the problem solving level, the strategy of verbal association requires a lower level thinking strategy. The coded thinking strategy involved subjects in the association of words from their native language with words in BASIC, rather than applying general principles to construct a computer program to solve the problem. This strategy tended to isolate the subject's concentration upon a single instruction in the problem statement, rather than focusing upon the problem as a whole or even logical partitions within the problem; thus, during the course of the present investigation, subjects partitioned the problem with respect to the statements in the problem description. As a result, the subjects tended to use a lower level thinking strategy, coded thinking, for analyzing the problem.

Debugging Strategies

When a program did not run correctly or stopped execution unexpectedly, the subjects employed debugging strategies to correct the problem. With BASIC programs, the computer halts execution of a program at the identification and point of a syntax error. When the computer runs the

program to its logical end and the program outputs incorrect results, a logic error is embedded somewhere in the program.

Syntax Error Strategies

Following approximately two months of instruction, the subjects did not have a clear understanding of the syntax of the BASIC language. This fact was obvious from the first observations, Day I, as the subjects experienced syntax errors during most computer runs of their programs and struggled to correct these errors throughout the greater part of the exercise period. At the second observation, Day II (i.e, about five weeks later), the subjects experienced fewer difficulties with syntax and generally did not have to ask the teacher for help with the correction of syntax errors. Rather, they were able to correct the syntax errors with confidence and efficiency. However, one characteristic that remained consistent from the first period to the next was that when the subjects experienced syntax errors, they worked sequentially to effect corrections.

Students were aided in the correction of syntax errors because of the nature of the BASIC language: that is, the computer program halted execution at the point of error. For the most part, subjects employed a guess-and-check strategy for correcting errors. They guessed at an error correction, entered the correction, and then re-ran the pro-

gram to determine if the error had been corrected. If the computer ran past the point of error, the subjects considered the error to have been corrected; if the computer halted at the same point, they attempted to change the form of the statement at the point the syntax error occurred, continuing this guess-and-check strategy until the error was corrected. If the subjects were unfamiliar with the error identified by the computer, or if they were unsuccessful with the guess-and-check strategy, the teacher was asked for help.

At best, the guess-and-check strategy is related to Gagné's (1970) concept learning, the part of the model which allows an individual to provide a response that identifies an entire class of objects or events. The similarity between concept learning and the guess-and-check strategy used to correct syntax errors can be seen more clearly through a particular example provided by Gagné (1985) as an explanation. A child, seven years of age, was shown a set of three hollow blocks, of which two of the blocks were identical. The experimenter followed an identical rule in always placing a candy reward under the non-similar block. The child was told that a small candy was under one of the three blocks. Initially, the child simply guessed, and lifted one or more blocks to find the candy. After one or more errors, the child chose the correct block and found the candy. The experiment was repeated using different sets of hollow blocks, but always retaining the odd block

as the source of the candy. After repeated exercises, the child recognized the odd block in each set and was able to perform the task repeatedly without error. In this example, the child used several trial-and-error exercises before learning the concept for identifying the block with the candy.

In the present study, students used trial-and-error methods to correct syntax error problems. The guess or the trial was correct if the program ran past the line which contained the syntax error. Otherwise, the guess was incorrect and the computer halted the program at the same place. Students continued to enter guesses for commands until the error was corrected.

At best, the learning processes of the subjects were confined to the concept learning level as they employed the guess-and-check strategy to correct syntax errors. During the first observation, the subjects clearly adopted the trial-and-error or guess-and-check strategies to learn concepts embedded within particular BASIC codes. During the second observation, the subjects seemed to have learned the concepts; they were able to recognize classes of particular errors and to fix those errors more quickly. Thus, even though the guess-and-check strategy was employed to correct syntax errors during both observations, differences in the application of the method were observed.

Wickelgren (1974) identified three types of trial-and-error strategies, providing an explanation for the differ-

ences. The first and least powerful strategy was called random trial-and-error, where the problem solver applied allowable operations to the givens within the problem. In a second, more powerful strategy, systematic trial-and-error, the problem solver created a scheme for systematically generating different sequences of actions which guaranteed that all possible sequences would be generated. And, in the third and most powerful strategy, classificatory trial-and-error, the problem solver organized sequences of actions into classes that were equivalent in some respects to the solution of the problem. If one action within the class resulted in the solution of the problem, then every sequence of actions grouped in that class would work to solve the problem.

In the present study, during the Day I observation, the guess-and-check strategy that the subjects used to attack syntax error problems resembled random trial-and-error. The subjects did not organize commands in a sequence or refer to classes of solutions. In actuality, students entered familiar commands that they thought would correct the syntax errors. If the result was not correct, they randomly tried a second approach. During the second observation on Day II, the guess-and-check strategy used to attack syntax error problems more closely resembled the second type trial-and-error approach, or systematic trial-and-error. The subjects were able to analyze errors and quickly respond with corrections, indicating the use of

systematic methods for correcting errors. Therefore, the subjects of the present investigation were not observed using the most powerful of the trial-and-error strategies identified by Wickelgrem (1974).

Ausubel (1968) identified two principle problem solving approaches, both of which occur at all age levels. The first approach is trial-and-error, consisting of random or systematic variation, approximation, and correction of responses until a successful variant emerges. The second is the insightful approach, which implies a "set" that is oriented towards the discovery of meaningful means-end relationships underlying problem solutions. Furthermore, this approach may involve either the simple transposition of a previously-learned principle to an analogous new situation, or a more fundamental cognitive restructuring and integration of prior and current experience to fit the demands of a designated goal.

The guess-and-check strategy used by the subjects of the current study is similar in manner to the trial-and-error strategy identified by Ausubel (1968). Ausubel noted that the trial-and-error or the guess-and-check approach is more or less inevitable in problems where no meaningful pattern or relationship exists or is discernible. Trial-and-error is generally characteristic of motor learning, wherein this practice is critical to the development and maintenance of motor skills. According to Ausubel, the trail-and-error or guess-and-check method is not as strong

a strategy as the insightful problem solving approach in which the learner engages in such activities as transforming information by analysis, synthesis, hypothesis-formulation, testing, rearrangement, recombination, translation, and integration.

Pea, Clement, and Mawbey (1986) used the term trial-and-error to identify the approach students used for debugging their program. It was suggested that this trial-and-error method neither engaged students in high level thinking skills, nor supported increased mastery of the programming language.

In summary, the guess-and-check strategy for correcting syntax errors observed during the present study is similar in nature to the Gagné (1970) concept learning. In addition, the guess-and-check strategy used during Day I was similar to random trial-and-error, and on Day II the approach was similar to systematic trail-and-error. However, concept learning does not imply learner involvement in high level thinking that can be equated with problem solving learning. Likewise, Wikelegren (1974) noted that random trial-and-error and systematic trial-and-error strategies were not as powerful strategies as the classificatory trial-and-error strategy. Ausubel (1968) indicated that trial-and-error was not as strong an approach as insightful problem solving. Finally, Pea et al. (1986) stated that trial-and-error strategy did not engage programmers in high level thinking skills nor support increased mastery of the

programming language. Accordingly, the guess-and-check strategy used by novice programmers during the present investigation did not demonstrate the high level thinking skills which characterizes problem solving.

The subjects used a second strategy to correct syntax errors; that is, they sought help from the teacher. The question is whether asking the teacher for help demonstrates low level or high level thinking? During the course of this investigation, the teacher provided directions for correcting the errors rather than asking the subjects to proposed methods of correction. Thus, the teacher analyzed the situation for the subjects, identified a correction, and the subjects simply responded to this information. Gagné (1985) explained problem solving as a method of learning that requires the learners to discover the higher-order rules without specific help. Presumably, the problem solvers will thus construct new rules idiosyncratically. During the current study, when students asked the teacher for help, they did not construct or discover a higher-order rule to solve the problems, and rather relied upon the teacher for help. Therefore, according to the Gagné definition of problem solving, asking the teacher for assistance cannot be considered a high level strategy.

Logic Error Strategies

Logic errors were more serious problems for the subjects of the present study since the computer did not iden-

tify the location of the errors. In this situation, the subjects did not apply a sophisticated debugging strategy, breaking the problem into small pieces and checking codes for each piece to trace the flow of the program. In fact, the subjects did not try to correct the logic errors by themselves; rather, they asked the teacher for help. The teacher identified a location in the program in which the error could be sought, and then provided assistance in correcting the error. With this assistance, the subjects used a guess-and-check strategy to debug logic errors. They guessed at changes to make in the program by considering the commands and statements with which they were familiar. They entered and executed changes until they identified codes that generated a correct result and provided a correct problem response.

During both the Day I and Day II observations, the guess-and-check strategy used to correct the logical errors was similar to Gagné (1968) concept learning as well as to the random trial-and-error method described by Wikelgren (1974) and trial-and-error as identified by Ausubel (1968). In logical errors situations, the computer did not specify error locations, and the subjects had to guess at the error locations as well as the commands required to correct the problem. The subjects did not organize the commands in a sequence order or into classes that were equivalent with respect to the problem solution. Rather, subjects entered the codes which with they were familiar in the hope that

they would correct logical problems. This technique was continued until the correct results were generated. This strategy for correcting the logic errors is not as powerful as a classificatory strategy.

Assistance from the teacher and the guess-and-check were the strategies the subjects used to attack logical errors, just as they had for syntax errors. However, the order of strategies was changed. With syntax errors, the subjects tried the guess-and-check strategy, then asked the teacher for assistance. For logical errors, the subjects first sought help from the teacher and then applied the guess-and-check strategy. As previously noted, these strategies are not based upon the high level thinking or analytic processes required for problem solving learning. Furthermore, Pea et al. (1986), Wickelgren (1974), and Ausubel (1968) stated that random trial-and-error methods did not constitute strong problem solving strategies. In addition, asking the teacher for help does not require students to think how the logical errors could be corrected. In fact, from the observations conducted for this study, the teacher virtually corrected logic errors for the subjects. Therefore, asking the teacher for assistance, followed by guess-and-check attempts at solution, does not constitute problem solving learning as defined by Gagné (1970).

Programming Process

In this investigation subjects used a similar process to solve the problem during both Day I and Day II observations. From the first observation to the second, the only identifiable difference in the process was the type of errors encountered within the program. On Day I, the subjects spent much of the period correcting syntax errors, whereas during Day II, the subjects spent most of their time correcting logic errors. Their lack of functional knowledge of the BASIC language seemed to divert the subjects concentration upon the whole problem toward concentration upon specific problem areas. As Pea and Kurland (1983) suggested, if students fail to learn the language in the first place, one should not expect them to gain in problem solving skills.

In general, the process which the subjects used to solve the problems began with reading the problem at the computer. They did not plan their programs in spending time away from the computer developing an algorithm or identifying a solution plan, then translating the plan into code and checking the code prior to statement entry into the computer. Rather, they read the problem and immediately began to enter coded statements at the computer. Dalby, Tourniarie, and Linn (1986) witnessed a similar strategy, the "rush-to-program" strategy which they indicated did not constitute a cognitive strategy. Glanter (1983) submitted that preplanning frequently occurred when

programming was done in Pascal (i.e., a compiled language), but that little or no planning was a frequent approach prior to writing codes for a programming language such as BASIC or Logo (i.e., interpreted languages). Moreover, Pea and Kurland (1984) noted that learning how to plan was not intrinsically guaranteed when working with interpreted languages.

In general, the subjects perceived the problem as consisting of several parts, each of which were identified by sentences or instructions in their order of appearance in the problem description. The problem situation was interpreted sequentially for each individual problem instruction. Upon successful completion of one instruction, the subjects advanced to the next instruction. As a result, the final program consisted of a sequence of statements which matched the problem situation.

Subjects entered codes that translated instructions in the problem specification to BASIC codes. When lines or sentences were reached that were not clearly translations of BASIC codes, the subjects guessed at the correct codes. Upon executing these codes, syntax errors were identified. Subjects then guessed at the correction of the syntax errors or asked the teacher for assistance. The teacher helped the subjects correct the syntax errors, then elimination of the syntax errors revealed logic errors. The first strategy selected by the subjects to handle these errors was to seek help the teacher. The teacher helped

then to locate the errors and provided explanatory corrections, following which the subjects returned to a guess-and-check strategy to debug the program. This guess-and-check strategy was often not successful and rather blocked the subjects from proceeding with the program, necessitating further requests for assistance from the teacher.

The question remains as to whether this process in which the subjects were involved was consistent with the problem solving process as identified by Polya (1988). Did the subjects learn to be problem solvers while learning to program? The data from this study do not support a positive response to this question, as defined by Polya.

Polya (1988) stated that the first stage of the problem solving process would consist of an attempt to understand the problem. Thus, the problem solver asks questions about the problem. What is known? What are the data? What is the problem situation? In this stage, the problem solver identifies the meaning of the key words, identifies the relevant data, and searches for relationships among the data in order to understand what is being asked. In the present study, the subjects did not engage in a similar process of problem understanding, and literally no planning of complete solutions appeared to have been undertaken. The subjects knew that the problem was to be solved using BASIC programming. Therefore, they immediately began to solve the problem by thinking in terms of BASIC codes.

Polya (1988) described the second stage of the problem solving process as an attempt to devise a plan. The problem solver thinks of similar problems previously solved. If the problem solver finds a similar problem or is able to solve simpler problems, then the problem solver must understand the method used to solve the related problem and apply it to solve the problem at hand. In the present study, the subjects did not apparently devise plans which encompassed similar problems previously solved. Rather, the subjects approached the problems sequentially line-by-line from the problem statement.

Polya (1988) described the third stage of the problem solving process as the attempt to carry out a plan. The problem solver checks each step of the plan to determine if the strategies function correctly. The problem solver must give consideration to the order of the plan and not lose sight of the connections between its major steps. The subjects for the present study did not create plans, much less follow them. Rather, the subjects dealt with the problem statement, and then with errors of either syntax or logic sequentially, as each was presented or occurred

Polya (1988) describes the fourth stage of the problem solving as a process of reevaluating the problem. The problem solver checks the results obtained to determine whether the outcomes are reasonable and correct. If possible, the problem solver extends and generalizes the solution to a similar situation by finding or creating up and

exploring a related problem. The subjects for the present study did not analyze the problem for variations in inputs/outputs. They simply accepted the program as correct if a correct output was received for one test.

The problem solving model proposed by Polya (1988)is similar to the models proposed by Kurlik and Rudnick (1988) and (Rubinsten 1975), in which the sequence of events involved in the problem resemble the process described by Dewey (1910). According to Dewey the initial event is problem presentation, which may be accomplished verbally or by other means. The problem solver then defines the problem, or distinguishes the essential features of the situation. As a third step, the learner formulates the hypotheses that could be applicable to a given situation. Finally, verification of the hypothesis, or successive verifications, is attempted until the learner obtains a verified solution result.

The programming process engaged in by the subjects of the current study was not entirely similar to the programming process described by Pea and Kurland (1983). The Pea and Kurland programming processes require the following steps: understanding the problem, designing and planning, coding, and using various debugging strategies. For the current investigation, subjects neither spent time in thinking about the problem, nor did they engage in planned solutions. The subjects programming processes began by reading the problem at the computer, immediately followed

by the entry of coded statements into the computer. Subjects entered codes that translated instructions in the problem specifications into BASIC codes. Upon successful completion of one instruction, the subjects advanced to the next instruction.

The programming process that students used for the current study was also not equivalent to the problem solving process identified by Krulik and Rudnick (1988) or by Rubinsten (1975). The problem solving process identified by Kurlik and Rudnick is based upon a sequence of steps which explore, select strategies, attempt solutions, and then reevaluate the problem. For the present study, students did not spend time exploring the problem; they did not select different strategies to obtain solutions, nor did they attempt to verify their answers. Their programming process started with the entry and execution of codes, and once these codes produced a correct result, each student proceeded to the next part of the problem.

The problem solving process identified by Rubinsten (1975) engages the problem solver in the following steps: preparation, incubation, inspiration, and verification. For the current study, the students did not seem to follow this type of problem solving process. They neither studied nor examined relationships between the elements of the problems posed, nor they did they spend time thinking about the problem. They did not check results against desired goals. Once the subjects were given the problem, they

moved immediately to the computer and engaged in the entry and execution of BASIC codes.

The programming processes described for this study were similar in nature to the model proposed by Dahmus (1970). This model involved students in a translation process in which each verbal statement was translated into a corresponding mathematical statement. The concrete translation of all facts is the key feature of this method. However, the order of the statements and their respective parts within the translation are expected to be identical to their expression in verbal form, that is, they are direct translations. For the current investigation, students entered codes that were translations of problem specification instructions into BASIC codes. The subjects thus perceived each problem as a set of parts, rather than as an integral whole and they attempted to translate from the problem specifications to BASIC codes in order to solve each part of the problem sequentially. However, research has suggested that the Polya (1988) problem solving process is more effective than the Dahmus problem solving model (Bassler, Beers, & Richardson, 1972; Gordan, 1977). Moreover, none of the research reviewed has noted that the Dahmus method is compatible with other problem solving processes reviewed.

Limitations of the Study

Several aspects of this study limit the representative nature of the findings reported, including the nature and/or composition of the sample, the computer programming language used to solve the problems, the teaching method, the number of times data were collected, analysis of the results with respect to a particular problem solving process (Polya, 1988), and the nature of the problem in the context of the length of time allowed for problem solution.

Only 13 students from a restricted geographical area were selected for the inclusion in this investigation. To further strengthen the generalizability of these findings, a larger number of students from different geographical areas would be required. In addition, this study purposely narrowed its focus to novice programmer thought processes while engaged in BASIC programming. BASIC is an interpreted language, whereas other languages such as PASCAL are not. Moreover, unlike Logo, BASIC is not a graphic-based language. Therefore, the findings of the present study do not necessarily represent the thought processes of individuals engaged in other programming languages and are thus confined only to novice programmers in the BASIC language.

The findings of this study were generated from two days of data collection during a single high school semester. Providing more than two days of data collection may vary the results to an unknown degree. Moreover, collecting data in the second semester where students have more instructions and experience in BASIC programming may also vary the results to unknown degrees.

In some instances the subjects did not complete their recorded remarks; that is, subjects, while speaking into the recorders suddenly became silent in mid-sentence or thought. Thus, the recorder was not at all times able to write complete descriptions of what the problem solver was thinking or doing at any given time making it difficult to clearly identify what the subject was thinking. In addition, the subjects resisted the instruction not to ask for help from the recorder. Some students cooperated with one another, resulting in an inability to distinguish between the voices of the problem solver and the recorder during the audiotape transcription process. In addition, only the final listing of the problem solver's program was available for analysis and not all of the changes effected in the program throughout the entire exercise period were available.

The current study was also limited by the use of a single comparative model of problem solving; that is, the Polya (1988) method. The problem solving processes used in such fields as psychology might provide a different per-

spective on student thought processes while engaged in the solution of computational problems.

Although the problem situations given to the subjects of the current study were identified as reasonable for the level of students which composed the sample, the situation posed may have been more like excerises than problems. According to Krulik and Rudnick (1988), exercises are situations that involve drill and practice to reinforce previously learned skills; problem resolutions are situations that require thought and the synthesis of previously learned knowledge. These problems developed for the current study and given to novice programmers may have been exercises in nature since the students were first introduced to such programming concepts as INPUT or looping, and then were asked to provide solutions for these problem situations. Moreover, these problem situations may have been more akin to exercises since the students were able to break the situations into sequential parts by sentences. These types of situations, characteristic of problems given to students in programming classes, may not encourage students to use strategies which may be used in problem solving. In other words, the students may have used the problems posed as drill and practice to reinforce previously learned programming concepts, rather than as a basis for the synthesis of previously learned programming knowledge for the purpose of problem solution.

The length of time the subjects were given to solve the problems may have affected the findings of this study. Subjects were allowed one-half hour to solve each problem, which may not have been sufficient time and may have caused the subjects to rush their solution approaches. In typical secondary school computer programming classes, students may be given an entire 50-minute period or longer to create a solution to a problem. Thus, it is possible that allowing the subjects more than 30 minutes to solve the problems could have varied the results of this study. Furthermore, in typical programming classes, students are given pre-described problems, rather than allowing them to create their own problems. Creating and solving their own problems may encourage them to discover more than one strategy to solve problems. Thus, the results could have differed if the students had been allowed to create and to solve their own problems.

For this investigation, since the teacher effect was not observed during periods of classroom instruction, the effect the teaching method had upon the findings cannot be known. No standard curriculum for computer science courses exists at the secondary level. Moreover, the state of Oregon has not adopted specific textbooks for computer science courses. In fact, the computer curriculum at each school is typically designed by the teacher; this means that the objectives, the instructional materials and types of problems, and the methods of evaluation are selected by

the teacher. For this reason, what students learn in computer programming classes is dependent to a greater degree upon the teacher than upon compliance with a set of statewide goals. The results of this study are also limited by the fact that subjects were selected from within a single class. Finally, the results of this investigation are limited by the lack of information about the effect of the specific teaching methods on student programming processes and problem solution strategies.

Recommendation for Future Research

This study observed student thought processes while engaged in BASIC language programming. Future research should examine student thought processes when working with a compiled language such as Pascal. Typically, programming in Pascal requires programmers to think about the whole program rather than line-by-line, as in BASIC. In Pascal, all syntax errors must be corrected before any part of the program can be executed to produce results, while in the BASIC language, the program can run partially even in the presence of syntax errors. In the BASIC language, the program runs sequentially until it is interrupted by syntax errors; as a result, students are able to receive a report of partial results. Consequently, an investigation of student thought processes while using a complied language may manifest a different pattern of thought processing.

A larger sample of students from diverse geographical areas is also recommended to obtain findings more representative of a general population. In addition, future research should investigate the thought processes of students who have had more experience than a single term of programming. It may be that novice programmers reflect different programming thought processes and strategies than students who have had additional instruction and experience in programming.

A case study of from two to three students explored over a longer period of time may provide a more clear description of student thought processes. Students could thus be observed while solving several problems during a term or year, both within in a classroom environment in which they have only limited time (i.e., 50 minutes) and outside the class with unlimited time for problem solution. Investigating student thought processes both inside and outside the class environment may demonstrate whether the time constraint has an effect upon student programming behaviors. Interviewing these students would also provide additional information about their approach to solving problems. Equally important, the teaching method and the nature of the class could be observed to determine if they have an effect upon student programming behaviors.

For the present study, students had the opportunity of either solving the problem at the computer or at their own desks. Students chose the computer and immediately entered

codes in order to solve the problems. Future studies may need to investigate whether the results would be different if students were required to initially write the program at their desk with limited computer time for program entry. Working at the desk before entering codes into the computer may force students to spend more time thinking about the whole problem. Based upon this model, students may be more likely to describe their thought processes while solving the problem.

Furthermore, future research may be needed to study the problems that are given to the students in computer programming courses. In computer science courses, students are usually given predescribed problems, rather than allowing them to design their own problems. If students create and solve their own problems, they may discover several problem solving approaches to solve the problem. Discovering these problem solving approaches may help students to gain problem solving skills.

Implications for Computer Science Education

The findings of this study did not support the rationale that students acquire problem solving skills in the process of learning computer programming. There may be values in teaching computer programming, values that meet the identified goals of schools. However, schools must investigate the value of teaching computer science in rela-

tionship to the values of teaching other subjects. It would not appear to be wise for schools or educational programs to emphasize computer programming instruction for the purpose of learning problem solving skills, if this learning takes place at the expense of other subjects. This is especially true in view of the cost factors involved, given the need for computer equipment and trained teachers. However, the computer science curriculum may offer additional values to meet these educational goals.

If the goal of teaching computer programming is to gain problem solving skills, educators may need to investigate the computer science course curriculum to emphasize specific strategies for creating computer programs. In addition, allowing students to create programs without the use of modular programming techniques encourages the view of a problem as a series of parts, rather than as a whole. The use of modular programming techniques may thus encourage learners to think about the problem holistically, while decomposing it into logical parts, rather than looking at the problem as a sequence of parts within a problem statement. Furthermore, breaking the problem into several parts may lead the programmer to new capabilities for the development of thought process about each separate part of the problem, while at the same time providing the opportunity to link these parts into an integral problem solution. With modular programming techniques, students are allowed to isolate specific parts to analyze logic and syntax

errors on their own without assistance from a teacher. Modular programming strategies may increase the novice programmer's ability to find and diagnose problems, and thus may encourage the learning and use of higher order problem solving principles, or sets of strategies that serve to determine the direction of thought processes.

Students may learn problem solving skills more effectively to the degree they are confronted with actual problems rather than exercises, that is, problems which allow students to explore and think about solutions without time constraints. Students should not be limited to 30 minutes to solve problems; rather, they should be given time for exploration. Involving students in such activities may provide the opportunity to think and synthesize previously learned programming concepts (Kurlik & Rudnick, 1988). In addition, a problem situation may encourage students to try different approaches in solving problems Accordingly, the teacher's task is to organize appropriate problem solving situations. Furthermore, teachers may want to provide students with problems that are novel, in the sense of presenting unfamiliar situations that are within students' capabilities, so that students can more readily apply previously learned skills and knowledge.

The current study described students engaged in a guess-and-check strategy, similar in nature to Gagné's (1970) concept learning, which is two levels below problem solving learning. Ideally, to gain problem solving skills,

rather than concept learning, student activities should be at the problem solving level. It is perhaps more likely that if students learn skills in computer programming similar to the skills learned at the problem solving level, these new skills may be moved to the problem solving level. Consequently, computer science educators may need to provide programming assignments that build on previously completed assignments, allowing students to provide meaningful links between their learning areas. These links may create a bridge between programming and problem solving and may encourage students to move from concept learning to problem solving learning.

The present study also found that students often rely upon the teacher for help in solving problems. When the teacher does most of the thinking, the question of what incentive the students had for correcting their own logic and syntax errors arises? Asking the teacher for help was a strategy for solving the problems that did not require a problem solving level strategy. Gagné (1970) described problem solving as a method of learning that requires the learners to discover higher-order results without specific help. In this manner they are more apt to construct new rules in their own idiosyncratic manner. Therefore, the role of the teaching must be to encourage the students to solve problems, rather than to rely upon the teaching staff for continuing assistance. The teacher must resist the temptation to guide the students toward a "correct" answer.

In essence, teachers presenting learners with solutions is a process which does not require the learners to generate higher-order rules, the learning that results from the application of problem solving techniques.

REFERENCES

Ausubel, D. P. (1968). *Education psychology: A cognitive view*. New York: Hold, Rinehart and Winston Inc.

Bassler, C. O., Beers, I. M., & Richardson, I. L. (1972). *Comparison of two instructional strategies for teaching the solution to verbal problems*. Nashville, TN: George Peabody College for Teachers.

Becker, H. (1983). *School uses of microcomputers*. Baltimore, MD: Johns Hopkins, Center for Social Organization of Schools.

Blume, G. W., & Schoen, H. L. (1988). Mathematical problem solving performance of eight-grade programmers and nonprogrammers. *Journal for Research in Mathematics Education, 19*, 142-156.

Bogdan, C. R., & Biklen, K. (1982). *Qualitative research for education*. Boston, MA: Allyn and Bacon, Inc.

Carpenter, P. C., Corbitt, K. C., Kepner, S. H., Lindquist, M. M., & Reys, E. K. (1980). NAEP note: Problem solving. *The Mathematics Teacher, 73*, 427-445.

Clement, C. A., Kurland, D. M., Mawby, R., & Pea, R D. (1986). Analogical Reasoning and Computer programming. *Educational Computer Research, 2*, 473-486.

Clements, H. D. (1986). Effects of Logo and CAI environment of cognition and creativity. *Journal of Educational Psychology, 78*, 309-318.

Clements, H. D., & Gullo, D. F. (1980). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*, 1051-1058.

Cordero, A. D. (1982). *The role of translation in second language acquisition*. Paper presented at the 97th Annual Convention of the Modern Language Association of America, Los Angeles, CA, December 27-30, 1982.

Dahmus, M. E. (1970). How to teach verbal problems. *School Science and Mathematics, 10*, 121-138.

Dalby, J., Tourniaire, F., & Linn, M. C. (1986). Making programming instructions cognitively demanding: An intervention study. *Journal of Research in Science Teaching, 23*, 427-436.

Dalton, D. W. (1986). *A comparison of the effects of Logo use and teacher-directed problem solving instruction on the problem solving skills, achievement, and attitudes of low, average, and high achieving junior high school learners.* Paper presented at the Annual Convention of the Association for Educational Communication and Technology. Las Vegas, NV.

Degelman, D., Free, J. U., Scarlato, M. P, Blackburn, J. M., & Golden, T. (1986). Concept learning in preschool children effects of a short-term Logo experience. *Journal of Educational Computing Research, 2*, 199-205.

Dewey, J. M. (1910). *How we think.* Boston: Heath.

Feuerstein, R. M. (1981). Mediated learning experience in the acquisition of kinesics. In B. L. Hoffer & R. N. St. Clair (Eds.), *Developmental kinesics: The emerging paradigm.* Baltimore, MD: University Park Press.

Feurzeig, W., Horwitz, P., & Nickerson, M. (1981). Microcomputers in education. Cambridge, MA: Belt, Beranek, and Newman, Report No. 4798.

Gagné, R. M. (1970). *The conditions of learning.* New York: Holt, Rinehart and Winston.

Gagné, R. M. (1985). *The conditions of learning and theory of instruction* (4th ed.). New York: Holt, Rinehart and Winston.

Gallini, J. (1984). A comparison of the effects Logo and CAI learning environment on skills acquisition. *Journal of Educational Computing Research, 3*, 461-477.

Glanter, E. (1983). *Kids and computers: The parents' microcomputer handbook.* New York: Putnam.

Gordan, M. (1977). *Prospective on problem solving: Person versus paradigm.* Paper presented at the Annual Meeting, American Educational Research Association. New York.

Hergenhahn, B. R. (1982). *An introduction to theories of learning* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall Inc.

Hillgard, R., & Bower, G. (1975). *Theories of learning.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

Horner, C. M., & Maddux, C. D. (1985). The effects of Logo on attributions toward success. *Computer in the Schools, 2*(2-3), 45-54.

Howell, R. D., Scott, P. B., & Diamond, J. (1987). The effects of instant "Logo" computer language on the cognitive development of very young children. *Journal of Educational Computing Research, 3*, 249-261.

Hunter, J. H., Kemp, T., & Hyslop, I. (1987). *Development and evaluation of the "thinking with Logo" curriculum.* Alberta, Canada: Dept Of Education.

Kransor, R., & Mitterer, O. (1984). Logo and the development of general problem solving skills. *The Alberta Journal of Educational Research, 30*, 133-44.

Krulik, S., & Rudnick, J. A. (1988). *Problem solving: A handbook for elementary school teachers.* Boston: Allyn and Bacon, Inc.

Kurland, D. M., Pea, R. D., Mawby, R., & Pea, D. R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research, 2*, 429-458.

LeBlanc, F. J. (1982). Teaching textbook story problems. *Arithmetic Teacher, 29*(6), 52-54.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classroom. *Educational Researcher, 14*(2), 14-29.

Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction. Instruction, access, and ability. *Educational Psychology, 20*, 191-206.

Markman, E. M. (1977). Realizing that you don't understand: A preliminary investigation. *Child Development, 48*, 986-992.

McCoy, L. P. (1986). *General variable skills, computer programming and mathematics.* Paper presented at the Annual Meeting of the International Association for Computing in Education, New Orleans, LA.

McGrath, D. (1988). Programming and problem solving: Will two languages do it? *Journal of Educational Computing Research, 4*, 467-487.

National Council of Supervisors of Mathematics. (1978). Position statement on basic skills. *The Mathematics Teacher, 71*, 147-52.

National Council of Teachers of Mathematics. (1989). *Agenda for action*. Reston, VA.

Newell, A., & Simon, H. A. (1972). Human problem solving. Englewood Cliffs, NJ: Prentice-Hall.

Oliva, P. F. (1969). *The teaching of foreign language*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Papert, S. (1980). *Mindstorm: Children Computers and powerful ideas*. New York: Basic Books.

Pea, R. D., & Kurland, D. M. (1983). *On the cognitive effects of learning computer programming* (Technical Report No. 9). New York: Bank Street College of Education, Center for Children and Technology.

Pea, R. D., & Kurland, D. M. (1984). *On the cognitive effects of learning computer programming* (Technical Report No. 16). New York: Bank Street College of Education, Center for Children and Technology.

Pintrich, P. R., Berger, C. F., & Stemmer, D. M. (1987). Students programming behavior in Pascal course. *Journal of Research in Science Teaching, 24*, 451-466.

Palumbo, D. B. (1990). Programming language/problem solving research: A review of relevant issues. *Review of Education Research, 60*, 65-89.

Polya, G. (1957). *How to solve it*. New York: Anchor.

Polya, G. (1988). *How to solve it* (2nd ed). Princeton, NJ: Princeton University Press.

Red, W. E. (1981). Problem solving and beginning engineering students. *Engineering Education, 72*, 167-170.

Rieber, L. P., & Lloyd. (1986). *The effects of Logo on young children*. Paper presented at the Annual Convention of the Association for Educational Communication and Technology. Las Vegas, NV.

Rosati, P. (1985). Experimental use of a computer problem solving tutorial in engineering static. *Journal of Educational Technology Systems, 14*(1), 15-21.

Rubinsten, M. F. (1975). *Patterns of problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Swan, K. (1989). *Programming objects to think with: Logo and the teaching and learning of problem solving.* Paper presented at the Annual Meeting of American Educational Research Association, San Francisco, CA.

Swan, K., & Black, J. B. (1988). *The cross-contextual to non-computer of problem solving strategies from Logo.* Annual Meeting of the American Educational Research Association, New Orleans, LA.

Turner, S. S., & Land, M. L. (1986). Cognitive effects of a Logo enriched mathematics program for middle school students. *Journal of Educational Computing Research, 4*, 443-451.

Wickelgren, W. A. (1974). *How to solve problems.* San Francisco: W. H. Freeman and Company.

APPENDICES

Appendix A

Letter of Transmittal

Fall, 1991

Dear parent:

Oregon State University is conducting a study on the thought processes students use while engaged in solving computer programming. This information will be used to improve instruction in computer programing classes and to advance knowledge in the field of computer science education.

It would be helpful if you would allow your son/daughter to participate in this study. Participants will be given computer programming problems to solve during one class period; they will use an audio tape recorder to record the processes they are using while solving problems. Participants will follow this procedure once during November and once during January. Additionally, all participants in the study will complete a background information questionnaire. All information will be maintain in confidence and will not affect the student's grade.

Please complete the form and return it to your teacher. Thank you for your assistance.

Sincerely,

Margaret L. Niess
Major Professor

Aqeel Ahmed
Graduate Student

Return this portion to your teacher

(Name of student

__ may participate in the study.

__ may not participate in the study.

Signed _____

Date _____

Appendix B

Subject Background Information

The purpose of the following questionnaire is to obtain data on students' backgrounds in mathematics and prior computer experience.

Student number: _____

Gender: Male__ Female__

Grade level: ____

1. Do you have a computer in your home? yes__ no__

2. Have you used a computer? yes__ no__

3. Describe the computer courses you have taken, i.e., computer applications.

4. Check all computer languages with which you have some experience.
   __ BASIC
   __ Pascal
   __ Logo
   __ Other

5. Have you taken computer classes other than BASIC? If so, which courses and when?

6. Check all the math course(s) you have <u>already taken</u>.
   __ General math
   __ Pre-algebra
   __ First Year Algebra
   __ Second Year Algebra
   __ Trigonometry
   __ Calculus
   __ Other (please describe)

7. Are you currently taking a math class? If so which one?

Appendix C

Subject Profiles

AJ

AJ is a male student in the tenth grade who has a computer at home. He has taken two computer classes: a computer applications class and Logo. His preparation in math stems from these classes: General Math, Pre-Algebra, and current enrollment in Algebra.

Once AJ was given the MONEY problem during Day I data collection, he went directly to the computer and wrote codes on paper. He wrote "10 HOME" as the first line in the program, followed by the codes to ask the customer to enter a name. AJ entered these codes into the computer and added more codes that would ask the customer to input the cost of item in dollars. At this time, AJ had spent five minutes on working on the problem.

AJ ran his program and received syntax errors because of misspelled commands. He edited the program to fix the syntax errors. Following that, he executed the program and again received syntax errors, but did not note why he experienced the errors. He continued to change codes in the program to fix the errors, but was unsuccessful. At this point, AJ had exhausted 15 minutes on solving the problem.

In the next 10 minutes AJ talked with his partner. Their discussion was not related to solving the problem. Next, AJ typed the codes to find the number of quarters, dimes, nickels, and pennies in the customer's change. His codes were logically incorrect and made no sense for the given problem. For instance, he entered codes to divide .25 by 100 and assigned the results to X. AJ spent about 25 minutes working on the problem. His program ran without syntax errors, but did not produce the correct results.

AJ was given the CARS problem during the second period of Day II data collection. He worked at the computer, where he typed "10 HOME," followed by codes for a random number algorithm to generate a number between 1 and 8 for 100 times. Here, AJ experienced a logic error since he was generating a random number for 100 times. He needed to generate random numbers until the number eight was

achieved. Next, AJ tried to determine the type of car by checking each random number. So far, AJ had spent five minutes in solving the problem.

After five minutes, AJ was still trying to find the type of car by checking each random number and pointing the name of the car. During this time, he entered codes to accumulate the number of the same type of cars passing by. He created seven counters, one for each type of car.

After 15 minutes, he entered the codes that would display the type of car and the number of times that car passed by but received incorrect result. Here, he asked the teacher for help. The teacher advised AJ to check the random number when it was eight then the it would be time to quit counting cars. He edited the program adding instructions to check for quitting time. This fix did not work because he was using a FOR-NEXT loop to generate a random number for 100. AJ had spent about 25 minutes to solve the problem. His program did not produce the correct result.

Ed

Ed is a male student in the tenth grade who owns a computer. Ed had taken one computer course in a computer application and had experience with Pascal programming. Ed's background in mathematics is derived from enrollment in General Mathematics, Pre-Algebra, Algebra I, and Algebra II.

When Ed was given the MONEY problem during the second period of Day I data collection, he went directly to the computer. He first read the problem and then entered "10 HOME" as the first line of his program, followed by the codes asking the customer to enter the cost of the item. At this point, Ed had spent five minutes typing codes to solve the problem.

Ed executed the program (i.e., the part of the program completed), but received incorrect results. He did not note why the results were incorrect. Following this, Ed was quiet for two minutes. So far Ed had spent 10 minutes working on the problem, trying to compute the number of coins in the customer's change.

Afterward, Ed entered and executed codes to calculate the number of dollars in the customer's change, but again obtained incorrect results. He skipped the dollars part to work on the quarters part, but did not succeed. Ed was stumped, and therefore asked the teacher for help. At this

point he had spent 15 minutes on the problem and was waiting for the teacher's assistance.

Ed was still guessing and trying different codes to find the customer's change, but he was getting nowhere on the problem. Following this, he was quiet for one minute before observing that he wanted to find out why his program was not computing the number of quarters in the customer's change. At this point, Ed had spent about 20 minutes working on solving the problem.

Next, the teacher explained to Ed how to compute the number of quarters. After this assistance, Ed was able to calculate the number of dollars and quarters in the change. Then he typed and executed the codes to compute the number of the dimes and the program still did not work properly. Ed almost got the nickel part working. At this time, Ed had spent 25 minutes in solving the problem and had almost solved the number of each type of coins in the customer's change.

During the last five minutes Ed had his program working correctly, except that the results were off by one penny. Again, he asked the teacher for help. The time period ended and Ed was still trying to find out why the results were off by one penny. It seemed that he did not count the number of pennies correctly. Ed had spent 30 minutes in solving the problem. His program was logically correct, but it was off by one penny.

Ed was given the CARS problem during the second period of Day II data collection. He worked at the computer desk, where he approached the problem by reading it and typing "10 HOME," followed by more codes for the random number algorithm. Afterwards, Ed entered codes (GOSUB-RETURN) to create a subroutine. In this subroutine, he determined the make of the cars and printed the results. Following that, Ed executed the program but received syntax errors from the random number generator algorithm and had to edit the program to fix the errors. He then executed the program to see how it looked on the screen, but he did not note whether or not his program was working properly. Next, he tried to figure out the type of cars and the number of times each car passed by.

After 10 minutes, Ed was using the DATA statement where the names of the cars were stored. Following that, he executed the program, but did not note whether he received a correct result. At this point, he had spent about 20 minutes in solving the problem.

Following that, he added more codes to print the names of the cars and the number of times each car passed by. Ed

then executed his program several times to check the output appearance after fixing the program, but it failed to generate the needed output. He did not note why his program did not generate the proper output. Ed then guessed, typed, and executed different codes until the program generated the desired output. Ed had spent about 30 minutes in solving the problem. His program was logically correct.

Frank

Frank is a male student in eleventh grade. He has used computers and his family owns one. Before Frank enrolled in this class, he took a computer applications class and has taken several mathematics classes, including General Math, or Algebra, first and second year Algebra, and Geometry. He is currently enrolled in Algebra II.

Frank worked on the KWH problem in the first period of Day I data collection. He attempted to solve the problem at the computer desk and spent the first five minutes reading the problem. After that, he typed "10 HOME," followed by codes that asked the user to input the kilowatt-hours usage. Frank then ran the program (i.e., a partial program), but received syntax errors. Later, he found that his editor was not working because he did not boot the system. Therefore, he had to reboot the system and retype the codes.

During the next five minutes, Frank tried to determine how to compute the charge for kilowatt-hours usage and the total charge, but received incorrect results. He tried different codes, but still did not obtain a correct outcome. He eventually asked the teacher for assistance in computing the charge for the kilowatt-hours used. After the teacher's explanation, Frank typed the NEW command, which caused the loss of his entire program since he had not previously saved it.

After 20 minutes passed, Frank reentered his program code. He entered the codes that would find the number of kilowatt-hours between zero and 300, from 301 to 1200, and usage greater than 1200 and then computed the charge. When he ran the program, he received incorrect results. Frank said that he knew why his results were not correct.

During the last five minutes, Frank noted that he was going to do the math part, but he did not explain why. Later, Frank executed the program, but the results continued to be incorrect. Frank spent 30 minutes in solving the problem. Although his program ran successfully with no

syntax errors, he did not obtain correct results since it included several logical errors.

Frank received the CARS problem during the second period of Day II data collection. After reading the problem, he asked for help from the teacher in determining the codes to work with the random algorithm. With this information, Frank entered "10 HOME," followed by the algorithm codes to generate a random number between 1 and 8. He also asked his recorder to see if the random number algorithm codes were correct. When he executed these codes he received syntax errors from the random number algorithm.

At the end of five minutes, Frank tried to determine the makes of cars. Furthermore, he entered the codes that accumulated the numbers for each make of car. Next, He entered the codes for quitting time (i.e., when the random number was equal to 8). He also entered codes to print the type of car and the number of times that type had passed. Upon program execution, he received syntax errors because of misspelled commands. Frank corrected the syntax errors. Furthermore, he added more codes to display a message on the screen that had the number of times the same type of car passed. After fixing these problems, he received correct results. After 15 minutes, his program was logically correct but did not include the option to allow the user to run the program again.

Joe

Joe is a male student in the eleventh grade who had computer experience and had taken a Logo course. Joe's mathematics preparation was derived from classes in General Math, Pre-Algebra, first-year Algebra, Geometry, and Algebra II.

Joe attempted to solve the MONEY problem during the second period of the Day I data collection. He worked at the computer desk, read the problem and then typed "10 HOME," followed by the codes to instruct the customer to enter the cost of the item using an INPUT statement. Furthermore, Joe added a check in the program to make certain that the customer entered the correct amount ($10 or less). Afterward, Joe computed the change for the customer.

After five minutes, Joe ran the program (i.e., the part of the program completed), but received a syntax error at the point that the amount entered was checked. He edited the codes to fix the syntax error and at this point, Jon indicated that the tricky part was to determine how many coins the customer would receive. He was silent for

one minute before asking the teacher for help. The teacher showed Joe an example to illustrate the procedure for finding the number of coins.

After 10 minutes, Joe used the INT function to find the number of coins, but received a syntax error. After fixing the syntax error, the program produced incorrect results because of a logical error in applying the INT function.

After 15 minutes, Joe said that he did not know how to round-off the customer's change to compute the number of the coins. Afterward, he was quiet for one minute before repeating that he could not apply the INT function command to eliminate the decimal points. Since Joe was stumped, he asked the teacher for help. The teacher gave him some hints to find the number of coins.

After 25 minutes, Joe was successful at figuring out the dollar bill. Once he figured out the dollar part, he worked the quarters part, then the dimes. Joe was still working when the audio tape ended. Joe spent a little more than 30 minutes working on his program. His program was logically correct.

Once Joe was given the ELECTION problem during the first period of Day II data collection, he began by typing "10 HOME," followed by the instructions that would display the candidate names on the screen. Joe was sidetracked, creating user friendly codes rather than the codes for solving the problem.

After five minutes, Joe was still creating codes to print each candidate's name on the screen. He then entered codes for the random number algorithm to generate a number between 1 and 3 for 500 times. He executed the code but found the random number generator was not working properly.

At the end of 15 minutes, Joe was still trying to determine how to distribute the votes randomly among the three candidates. He was stumped and did not know what to do. He tried to determine the winner of the election, but could not do so. Therefore, he asked the teacher for help. The teacher explained how to distribute the votes randomly among the three candidates. Following that, Joe began to enter codes to distribute the votes among the three candidates and to determine who was the winner. To this point, Joe had spent about 20 minutes in solving the problem.

Joe executed his program but received incorrect results, noting that he did not know what to do or what to type. Next, Joe also noted that his random number algorithm worked correctly, but that he did not how to add the numbers of votes for each candidate. He asked the teacher

for help. The teacher gave him hints to find out how to accumulate the votes for each candidate. Joe had several syntax errors in his program and spent about 30 minutes solving the problem. Joe's program still contained syntax errors and thus failed to run because of these errors.

Jon

Jon is a male student in eleventh grade who owns a computer. Jon had taken a computer application class before enrolling in this class. His background in mathematics was derived from General Math, Pre-Algebra, Algebra, Geometry, and current enrollment in Algebra II.

Jon worked by himself attempting the KWH problem during the first period of Day I data collection. He went directly to the computer, where he spent the first five minutes reading problem one. In the second five minutes, Jon typed "10 HOME" as the first line in the program, followed by the instructions that would display the message on the screen: "THIS IS A PROGRAM TELLING HOW MUCH YOU WILL BE BILLING FROM USING ELECTRICITY." Here, Jon was sidetracked by typing user friendly codes that had no effect upon the solution of the problem.

After 10 minutes, Jon executed the codes that he had typed earlier, but did not note whether or not there were syntax errors in these codes. He then added an INPUT statement to his program. After 15 minutes, Jon attempted to determine the kilowatt-hours usage between 0 and 300 and between 301 and 1200 hours, and then to compute the charge. Logic problems resulted because he did not add the basic charge to the total charge. Also, for kilowatt-hours usage between 301 and 1200, he did not multiply the first 300 kilowatt-hours by .04237 and the remaining kilowatt-hours by .05241. Rather, he multiplied the entire number of kilowatt-hours usage by 0.05241.

After 20 minutes, Jon entered codes to find the kilowatt-hours usage greater than 1200 hours and computed the charge, but did not receive correct results. Again, he did not multiply the first 300 kilowatt hours by .04237, the next 900 kilowatt-hours by .05241, and the remainder by .06113; rather, he multiplied the entire values for kilowatt-hours usage by .06113.

During the last five minutes Jon still failed to obtain the correct results. He attempted to break the number of kilowatt-hours into three segments, and then to compute the charge for these segments. He was quiet while

entering the codes. After 30 minutes, although his program was running, the solution remained incorrect.

During the second period of Day II data collection, Jon worked on the CARS problem. Again, he worked at the computer, read the problem, and typed "10 HOME" as the first line in the program. He then entered instructions to display two messages on the screen. The first message was "WELCOME TO 10 DOLLAR STORE AND UNDER" and the second message was "HOW MUCH YOUR ITEM?" As for the KWH problem, he was sidetracked entering user friendly codes that did not effect the solution of the program.

After five minutes, Jon typed codes to accept the amount of money paid by the customer for the bill. To calculate the customer's change, he checked to determine if the amount paid was equal to or less than 10 dollars. He then entered the codes to print the number of dollars, quarters, dimes, nickels, and pennies in the customer's change. This effort was logically incorrect because the change had not been computed at that point.

After ten minutes, Jon said that he was confused and did not know how to figure out the change. He also added that he did not know how to do this problem at all. Jon was stumped, had no method of solving the problem, and spent only about 15 minutes solving the problem and was then unable to solve it. His program ran, but not did not return correct results.

Jon received the ELECTION problem during the first period of Day II data collection. He read the problem at the computer and then typed "10 HOME," followed by additional codes to display the candidates' names on the screen. When Jon executed the code, he received syntax errors because of misspelled commands. He fixed each command and then tried to enter codes for the random number algorithm to generate a number between 1 and 3 for 500 times. He indicated that he was having difficulty figuring out these codes; he erased some codes from his program without noting the reasons.

After 10 minutes, Jon was still working on codes for the random number algorithm. He did enter instructions to count the number of votes for each candidate. He deleted some codes, added others, but did not note the reasons whey these actions were taken. The codes did not produce correct results. Jon then guessed, typed, and executed different codes several times, but still obtained an incorrect outcome. At this point, Jon had taken about 20 minutes to attempt to solve the problem.

Next, he tried to write a subroutine, but he was unsure of the GOSUB-RETURN command. Thus, he asked the teacher for help. The teacher explained the use of the GOSUB-RETURN command. After receiving help, Jon deleted some codes from his program and created a subroutine to count the number of votes for each candidate. He executed the program, but received a syntax error since he forgot to insert a RETURN. On conclusion of the period, Jon's program ran but without producing correct results.

Ken

Ken is a male student in twelfth who had used computers; however, this class was his first formal computer class. His mathematics preparation included these General Math, Pre-Algebra, Algebra, Algebra II, Trigonometry, and current enrollment in Calculus.

Ken was given the MONEY problem for the second period of Day I data collection. He approached the problem by typing "10 HOME," followed by the instructions asking the customer to enter the cost of the item using an INPUT statement. Afterward, Ken computed the customer's change and tried to determine the number of the quarters in the change.

After five minutes of effort, Ken was experimenting with rounding the change to two decimal points. Then he executed the program and received syntax errors. Ken edited the INT function and executed the program, but still experienced syntax errors. Ken continued to try to fix the INT function command and continued to have problems with his experiments over the next ten minutes. Ken got the INT function command to work, but the results were incorrect. Next, Ken worked on the quarters part before computing the number of dollars. He ran the program, but obtained an incorrect outcome. Then, he worked on paper to figure out the number of quarters. At this point, Ken had spent 15 minutes solving the problem.

Ken then entered and executed the codes to find the number of quarters, but his program produced the wrong output. Therefore, Ken tried to find the number of quarters by guessing, typing, and executing different codes, but without any difference in the results. Ken appeared to type any instruction that came to mind. Later, he asked the teacher for assistance. The teacher explained how to use the INT function command to determine the number of quarters.

After 20 minutes, the teacher was still helping Ken, who edited his program to correct the change by deleting

lines, but without noting why he took the actions he did. Ken entered and executed the codes to compute the number of dollars in the change and eventually Ken's program generated the correct results.

In the last five minutes Ken asked the teacher if he was on the right track. After that Ken typed without speaking for about one minute before noting that he was working on computing the dimes. Next, Ken asked the teacher for more help to determine quantities for the other coins. Ken spent about 30 minutes solving the problem. His program was logically correct, but he did not compute the number of the pennies in the change.

During Day II data collection, Ken worked without a partner. He worked on the CARS problem at the computer desk. He typed "5 HOME," followed by the instructions to display the message "NUMBER AND KINDS OF CARS THAT DROVE BY." Next, he typed the codes for the algorithm to generate random numbers from 1 to 8 for 500 times. Here, Ken experienced logical errors since the problem did not ask for the generation of random numbers 500 times. In fact, the problem asked to check the random number when eight was obtained, indicating that it was time to quit. Afterward, Ken checked each random number to determine the kind of car.

After 10 minutes had passed, Ken added codes for a FOR-NEXT loop, executed the program but without producing the correct results. He edited the program to generate a random number for 100 times, executed the program, and again failed to obtain a correct output. He noted that his program was providing a cumulative total for the same makes of cars; thus, Ken had logical problems with the car counters. Next, Ken edited his program to display the name of each car and the number of times that this make had passed. Upon execution, he received the wrong results because the codes were not set to count each make of car. To this point, Ken had spent about 15 minutes solving the problem.

Ken executed his program and received incorrect results. At this time he was stumped and asked the teacher for help. The teacher advised Ken to determine if the random number was equal to 8, when it would be time to quit. The teacher also told him to take out the FOR-NEXT loop and use GOTO. Ken then entered the codes that display the name of each car as it passed. He executed his program, but received the wrong output because he did not count the other makes of cars (i.e., the types of car not included in this problem).

After 20 minutes, Ken's program ran, but the random number algorithm did not function properly. He edited and fixed the problem and the desired results were generated.

Ken had spent about 25 minutes solving the problem. His program was logically correct, but he did not include the option that allowed the user to run the program again.

Lee

Lee is a male student in the tenth grade who owns a computer. He took a Logo class before enrolling in this class. His mathematics backgrounds consists of courses in General Math, Pre-Algebra, first-year Algebra, and current enrollment in Geometry.

Once Lee had received the KWH problem during the first period of Day I data collection, he moved to the computer. Lee immediately asked the teacher for assistance. The teacher asked him if he had read the problem twice and to describe what he did not understand. The teacher gave him help as follows: "If you use between 0 to 300 kilowatt-hours, the rate is roughly four cents per kilowatt-hour plus the basic charge; if you use between 301 to 1200 kilowatt-hours, then the rate is about five cents plus the basic charge; and if you use over 1200 kilowatt-hours, the rate is about six cents."

After five minutes, Lee wrote his first line "10 HOME," followed by the instruction that would ask the user to input the kilowatt-hours usage. Following that, Lee tried to calculate the charge for the kilowatt-hours usage between 0 and 300 and between 301 and 1200 hours; Lee then noted that he had no idea how to solve the problem. At this point about 10 minutes were spent in solving the problem.

Lee received help from the teacher to determine which codes could be used to convert from kilowatt-hours usage to the total charge. Once he was able to compute the total charge for kilowatt-hour usage, he attempted to round off the total charge by multiplying it by 100, then dividing by 100. His method was incorrect since multiplying and dividing the total charge by 100 produced an identical net result. Lee guessed, typed, and executed different commands to round off the total charge, but without success.

After 15 minutes, Lee asked the teacher about the rounding INT function. The teacher told him to put the INT function in front of the total charge in order to round it off. Following that, Lee entered and executed the codes, but received syntax errors because of misspelled commands.

In the last five minutes, he ran his program several times, still continuing to experience syntax errors. Lee continued to have problems with the syntax of the INT

function. With additional help from the teacher, he was able to fix the problem. After 25 minutes, Lee had the program running with a logically correct solution.

Once Lee was given the ELECTION problem during the first period of Day II data collection, he went directly to computer where he read the problem and started to write codes. He wrote "10 HOME," followed by codes to display a message on the screen. The message had each candidate's name. Afterwards, Lee wrote the codes for the random number algorithm to generate a number from 1 to 3 for 500 times. Afterward, Lee tried to determine how to distribute votes randomly among the candidates, but was not able to do this successfully. At this point, Lee was stumped and asked the teacher for help. The teacher explained to Lee how to assign the votes randomly among the three candidates. Lee then typed and executed the codes, but received incorrect results. He edited his program to fix the counters for the number of votes for each candidate. He executed the program and continued to receive incorrect results since the counters still did not work properly. Lee had to edit the program to fix these counters.

After 10 minutes, Lee executed the program, but again obtained incorrect results. Lee guessed, typed, and executed different codes to fix the counters, but was not successful in fixing the problem. At this point, Lee asked the teacher for assistance. The teacher helped him count the number of votes for each candidate, following which Lee was able to fix the counters.

After 15 minutes, Lee obtained the correct output. He then added the option that allowed the user to repeat the election process; however, his option contained syntax errors since he compared string values to a numeric variable. After he replaced the numeric variable with a string variable, the program produced the desired outcome. In all, Lee devoted about 20 minutes to problem solution and finished with a logically correct program.

Mark

Mark is a male student in tenth grade who has a computer at home. Mark's prior experience with computers was derived from a Logo class and a seventh grade computer class. His background in math includes General Math, Pre-Algebra, first-year Algebra, and current enrollment in Geometry.

Mark was given the KWH problem during the first period of Day I data collection. First, he read the problem and typed "10 HOME," followed by instructions that asked the

user to enter kilowatt-hours usage using INPUT. Mark then executed these codes and received a message saying that an illegal command was typed, but he did not observe why this had occurred. Mark edited the program to fix the error. To this point, Mark had spent five minutes solving the problem, typing a few codes and executing them.

Mark then worked on the codes to find the number of kilowatt-hours between 0 and 300, 301 and 1200, and greater than 1200, and the total charge for the number of kilowatt-hours. Furthermore, he rounded the total charge and printed it. At this time Mark had worked 10 minutes typing codes to solve the problem piece-by-piece. Mark received syntax errors when he ran the program. He fixed the errors and reran the program. This pattern continued, owing specifically to a missing colon or parenthesis.

After 15 minutes, Mark was able to get part of his program running without syntax errors, but the results were incorrect. He did not note why the results were incorrect, but guessed, typed, and executed different codes to correct the problem. Mark edited the program and changed the total charge variable name from X to T. He executed his program, but did not obtain a correct result. Thus, he asked the teacher for assistance in rounding off the total charge to two decimal points. With the teacher's help, Mark was able to determine how to round off the total charge to two decimal points.

In the final five minutes, Mark's program reflected two syntax errors. He changed the plus sign (+) to an equal sign (=) in the round off statement. This fix did not correct the logical error of failing to divide kilowatt-hour usage into different segments; his program only instructed the computer to multiply the whole kilowatt-hour usage by one rate. Following that, he added an END statement to the program. After 30 minutes, he was able to correct the syntax errors, but his program did not generate the correct output.

Mark received the ElECTION problem during the first period of Day II data collection. He approached the problem by entering "10 HOME," followed by instructions that would allow the user to start the election. He then generated a random number from 1 to 3 for 500 times and assigned the votes randomly among the candidates. Furthermore, Mark entered the codes for the three counters to count the vote belonging to each candidate. To this point, he had spent five minutes solving the problem.

During the next five minutes, Mark entered the codes to display each candidate's name on the screen and the number of votes. He tried to find the winner and to dis-

play the winner's name on the screen with the number of votes. When he executed the program, there were syntax errors in the random number algorithm. Mark successfully, fixed the error. In all, he spent about 15 minutes solving the problem. His program was logically correct, but did not include the option to allow the user to repeat the election process.

Rick

Rick is a male student in the eleventh grade who has a computer at home. He took a Logo class before enrolling in this class. His mathematics background is derived from courses in General Math, Pre-Algebra, Algebra I, Geometry, and current enrollment in Algebra II.

Rick was given the KWH problem during the first period of Day I data collection. He approached the problem by typing "10 HOME," followed by instructions that would ask the customer to enter the kilowatt-hour usage using an INPUT statement. Later, Rick entered additional codes to instruct the computer to find kilowatt-hours usage between 0 and 300 and between 301 and 1200 hours, then to compute the total charge. Rick also added the basic fee ($3.00) to the total charge. So far, he had spent five minutes solving the problem and had completed a part of the program.

During the following five minutes, Rick continued to work on determining the total charge. He entered the codes to find the kilowatt-hours usage that were greater than 1200, calculated the total charge, and then added the basic charge ($3.00). He ran this part of the program, but received a syntax error because of misspelled code. He was able to fix the syntax error. At 10 minutes, he added more lines to the program and printed the results. When he ran the program, he received an unidentified syntax error. He corrected syntax error, but obtained incorrect results. Then Rick removed the syntax error, only to discover he also had errors in logic.

At the end of 15 minutes, Rick's program had produced incorrect results. Although he added the basic charge, the results were expressed in too many decimal places. He then worked with the rounding INT function, but without success. Rick struggled with the logical concept of rounding in BASIC. In the final five minutes the teacher assisted Rick in figuring out how to round off the results. With this assistance, he edited the codes and added a basic charge to the total charge for kilowatt-hours usage between 310 and 1200. After this fix, he still had an incorrect result and asked the teacher for help. Again the teacher explained the use of INT function command to round off the results.

With this help he was able to apply the INT function command correctly; he then ran the program and received the desired outcomes. Rick had spent about 25 minutes solving the problem. His program was logically correct.

Rick was given the CARS problem for the second period of Day II data collection. He approached the problem by typing "10 HOME," followed by the instructions to display the message, "CARS STATISTICS." Afterward, Rick entered the codes for the algorithm to generate a random number between 1 to 8. These numbers represent the makes of the cars. He identified the makes according to the random numbers.

After five minutes of work, Rick executed his the program but had misspelled some of the codes. He was able to fix these syntax errors quickly. Next, he entered the codes to print the makes of the cars and the number of times each make had passed. In the final ten minutes, Rick executed the program, but received syntax errors in the PRINT statements. Rick asked the teacher to check his program. The teacher responded that the program looked good. Next, Rick executed the program and obtained the desired outcome. Rick had spent about 20 minutes solving the problem. Rick's program was logically correct, but it did not include the option to allow the user to rerun the program.

Sam

Sam is a male student in eleventh grade and owns a computer. This class was his first formal computer class. Sam took several mathematics classes, including General Math, Pre-Algebra, first-year Algebra, second-year Algebra, and is currently enrolled in Algebra II.

Sam worked on the MONEY problem during the second period of Day I data collection. He first read the problem and then wrote "10 HOME," followed by instructions that asked the customer to enter the cost of the item using an INPUT statement. Again, Sam read the problem and wrote additional instructions to compute the customer's change and the number of dollars in the customer's change. Next, he typed all of the instructions into the computer.

After five minutes had passed, Sam was quiet for two minutes before asking the teacher for help. Because of the background noise, it was not clear what Sam asked. Sam stated that he was going to divide the change due by dollars, but his answer reflected the same amount of change. He entered different instructions in order to find the number of coins, but still failed to obtain correct re-

sults. Clearly, he had problems computing the number of dollars in change.

For 15 minutes Sam continued to try to find the number of dollars in the change. Even though Sam guessed and checked different codes, he still was unable to get the number of dollars. He stated that he was getting nowhere. For the last five minutes, the teacher explained to Sam how to used the INT function to find the number of dollars in the customer's change. Sam then seemed to know how to figure the dollars part. Once he computed the number of dollars, he tried to find the number of quarters, but time expired. Sam had spent about 25 minutes solving the problem. He had only the change amount and the dollar part logically correct. His program was incomplete because he did not compute the number of quarters, dimes, nickels, and pennies.

Once Sam was given the ELECTION problem during the first period of Day II data collection, he went to the computer and read the problem. Following that he typed "10 HOME," followed by the instruction that would display the candidate names on the screen. Next, he tried to type the codes for the random number generator. He had to ask the teacher for help in figuring out the codes for the random number algorithm. He entered the codes that would generate a random number 500 times, where each random number represented a vote. At this point, he had spent about 10 minutes solving the problem.

Sam seemed stumped and tried to determine how to distribute votes randomly among the candidates. He added instructions to display each candidate's name on the screen. Sam was then apparently sidetracked entering friendly codes, rather than working on the logic of the program. He then executed the program, but received syntax errors from the random number algorithm.

After 20 minutes had passed, Sam executed the program and received an incorrect result. Therefore, he guessed, typed, and executed different codes with no change in outcome. In fact, he had several logical errors since he had three random number algorithms and three loops in his program. Sam did not assign the votes randomly among the three candidates. At this point, he asked the teacher for help. The teacher helped him assign the votes randomly among the candidates. Sam was editing his program when the time was over. He had spent about 30 minutes solving the problem. His program did not run because of the syntax errors.

Sue

Sue is a female student in the twelfth grade. She had previously used computers in a 7th grade applications computer class. Sue had taken several mathematic classes, including General Math, Pre-Algebra, first-year Algebra, second-year Algebra, Trigonometry, and Geometry.

Sue attempted to solve the KWH problem during the first period of Day I data collection. Working at the computer desk, she read the problem and then typed "10 HOME," followed by instructions that asked the user to enter the kilowatt-hours used. She noted that she was confused, so she read the problem again. Sue also remarked that she knew what to do, but she could not put it into the computer. She then asked the teacher to check to see if she was on the right track. The teacher responded, "Yes." Sue typed only two lines and then asked the teacher to check her work. At this time Sue had spent five minutes solving the problem.

Next, Sue asked the teacher if she could break the problem into several parts. The teacher suggested that she read the example provided with the problem. Sue then entered the codes that would find and compute the charge for the number of kilowatt-hours between 0 and 300. She also asked her partner if she was on the right track. It seemed that she was not confident in her work.

After 10 minutes had passed, the teacher told Sue to add the basic charge ($3.00), but Sue responded that she did not know how to add it. The teacher advised her to break the number of kilowatt-hours into different parts, computing the charge for each part and then adding them together to obtain the total charge. Again, the teacher asked Sue to look at the example provided with the problem and try to solve it on paper.

After 15 minutes, Sue was still receiving help from the teacher in calculating the charge for the kilowatt-hours usage. The teacher helped her compute the charge for the first 300 kilowatt-hours, the next batch from 301 to 1200, and the last batch from 1201 and greater. Sue tried to print the total charge. She received syntax errors because she did not put an equal sign in front of the total charge equation. After 20 minutes, her program still contained syntax errors. Sue could not fix the syntax errors and again asked the teacher for help. She then corrected the syntax errors, but obtained incorrect results. Sue noted that she needed to round off the results. During the last five minutes, the teacher helped Sue add the basic charge. However, she was not able to correct the

syntax and the logic errors; she did not obtain the correct results.

Sue was given the CARS problem during the second period of Day II data collection. Sue started at the computer, entering "10 HOME" as the first line in the program. Next, she tried to figure out the codes for the random number algorithm and admitted that she did not know what to do. Her partner helped her to figure out the codes for the random number algorithm. She used the IF-THEN statement to identify the name of the cars that had passed, based upon random numbers. Sue did not understand how to instruct the computer to quit generating random numbers, so she again asked her partner for help. Her partner helped her determine the codes for the quitting time.

During the next five minutes, Sue tried to figure out the codes to count the number of times each car had passed, but experienced problems. Sue continued to ask the recorder for help in figuring out the codes for the counters. Finally, she typed the codes for seven counters (one for each number generated) in her program; these counters accumulated the number of times each car had passed.

After 10 minutes, Sue concentrated on codes to display the results on the screen. Her partner also helped in figuring out the codes for displaying the results. Afterward, she executed the program, but obtained a result of zero. She guessed, typed, and executed different codes and continued to produce incorrect results. Sue was stumped and asked her partner for help. After 15 minutes she still had incorrect result because the counters were not working properly. Sue checked the codes for these counters; her partner also helped check the codes for the counters and told her that she was using the same variable for more than one counter. Sue fixed the problem by using one variable for each counter and then received correct results. Next, she entered the codes that added an option to the program to allow the user to run the program again. With this addition, she received a syntax error. After editing the codes, her program ran and generated the correct output.

Tom

Tom is a male student in the twelfth grade and has a computer at home. Although he had used computers prior to this class, he had never taken a computer class prior to his current enrollment. On other hand, he had taken several math classes, including General Math, Pre-Algebra, first-year Algebra, second-year Algebra, and Geometry.

Tom was given the KWH problem during the first period of Day II data collection. He worked at his desk, where he wrote his program on paper. His first line was "10 HOME," followed by instructions to ask the user to enter the numbers for kilowatt-hours usage. For this code he used the PRINT statement to accept the number of kilowatt-hours from the keyboard, rather than from the INPUT statement (i.e., the proper command for this situation). Tom then wrote additional code to identify the number of kilowatt-hours between 0 and 300. At this point, Tom had spent five minutes just writing codes on paper.

Following that, Tom wrote the codes that printed the rate rather than the charge. In fact, Tom needed to compute the charge and then print it. Furthermore, he wrote the codes that would locate the number of kilowatt-hours between 301 and 1200 using IF-THEN statements. He incorrectly compared string variable (A$) with a numeric variable in the IF-THEN statement, identifying the number of kilowatt-hours used. Tom was confused about the syntax of the numeric and string variables.

At the end of 10 minutes, Tom tried to compute the charge for kilowatt-hours usage greater than 1200. He added the basic charge ($3.00) to the rate instead of adding it to the total charge. Tom experienced two logical problems: First, he added the basic charge to the rate, and second, he printed the total charge without prior computation. After 15 minutes of effort, he had written the whole program on paper, moved to the computer and entered the codes.

When Tom ran his program he received a syntax error because of the comparison of a string to a numeric variable. He became frustrated and was unable to fix the syntax error. Thus, he asked the teacher for assistance. The teacher advised him to use numeric variables, rather than string variables, to represent numeric values. With this assistance, Tom was able to fix the syntax error.

After 20 minutes, Tom executed the program without syntax errors; however, the program did not produce correct results. Tom edited his program, changing the PRINT statement to an INPUT statement, to ask the user to enter the number for kilowatt-hours usage. With this correction, the program still output zero. He continued to guess and to enter different codes, but the results remained incorrect. Again, Tom resorted to asking the teacher for assistance.

During final five minutes, Tom received help from the teacher. The teacher told him to multiply the number of the kilowatt-hours by the rate and then add the basic charge. Accordingly, Tom edited his program to add codes

that would calculate the charge. The time expired while Tom was still editing his program. After 30 minutes, Tom's program was running, but did not produce the correct results.

Tom was given the ELECTION problem during the first period of Day II data collection. He first attacked the problem by writing codes for the random number algorithm. He said that he had forgotten the codes for the algorithm to generate a random number. Tom read the problem, attempting to understand it more clearly. He entered codes for the algorithm to generate numbers between 1 and 3, but did not note whether or not these codes worked properly. Next, he entered new codes to display the candidates' names on the screen. At this point he attempted to identify a method of assigning votes randomly.

After 10 minutes, he tried to format the output of the program. He executed the program, but received syntax errors for the random number generator codes. After fixing the errors, he tried to print each candidate's name with the number of votes received. He experienced logic errors since he printed the number of votes for each candidate before computing them.

After 15 minutes, Tom continued attacking the problem of displaying candidates' names on the screen, placing the numbers of votes under each candidate's name. Again, Tom tried to print the number of votes for each candidate before assigning the votes randomly and counting these votes. Clearly, he had problems solving this problem; he did not provide a copy of his program to check its logic, although he had spent about 20 minutes on the problem.

Tony

Tony is a male student in the tenth grade who owns a computer. He took a Logo course before enrolling in this class. His mathematics background includes General Math, Pre-Algebra, first-year Algebra, and current enrollment in advanced Algebra.

Tony attempted to solve the MONEY problem during the second period of Day II data collection. He approached the problem by typing "10 HOME," followed by instructions to display the message "TEN DOLLARS STORE" on the screen. Afterward, Tony entered codes asking the customer to input the cost of the item and checked whether the customer entered the proper amount (at ten dollars or less). Tony also computed the customer's change.

After five minutes had passed, Tony executed the program (i.e., a part of the program code), but received a syntax error at the point where he checked to make certain that the proper amount had been entered. Tony did not state why he experienced the syntax error. At 10 minutes, Tony ran the program (or a part of the program) with no syntax errors, but still failed to obtain correct results. He tried different codes to determine the correct change, following which he executed his program and received a syntax error because of misspelled code. After fixing the syntax error, he ran the program but did not obtain the correct results. It seemed that Tony was stumped. To this point, Tony had spent about 15 minutes solving the problem.

Tony asked the teacher for assistance. The teacher suggested that he use the INT function to find the number of coins. The teacher told Tony, "Once you find the number of quarter(s) in the change, then you need to subtract the number of quarters from the change to find the rest of the coins." Furthermore, the teacher demonstrated an example that illustrated the method of finding the number of coins. Tony had worked on the problem for 25 minutes.

With the teacher's help, Tony was able to proceed. Tony entered the codes to figure out the number of coins in the customer's change. The audio tape ended as Tony was still typing codes. Tony had spent a little over 30 minutes solving the problem. His completed program was logically correct.

Tony received the ELECTION problem during the first period of Day II data collection. He read the problem and then typed "10 HOME," followed by instructions that would display this message, "ELECTION DAY AND VOTERS ARE COMING." Then, he entered the codes to generate 500 random numbers between 1 and 3, following which he worked to distribute the votes randomly among the three candidates. Afterward, he executed these codes and received a syntax error, but did not note why this occurred. He edited the program to fix the syntax error and changed the codes to generate 500 random numbers in place of 512.

After five minutes, Tony typed but said nothing for about five minutes. Then he mentioned he had difficulty distributing the votes randomly because of the number of votes for each candidate. He attempted to determine which candidate had the largest number of votes. To this point, Tony had spent about 15 minutes solving the problem.

During the final 10 minutes, Tony deleted some lines from the program and then added new code lines, without stating why this was done. He then executed the program and received the correct results.