1992

Effectiveness of training on algorithms versus notation for indirect addressing comprehension

Robert Dee Franks
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons, Educational Assessment, Evaluation, and Research Commons, Experimental Analysis of Behavior Commons, Psychiatry and Psychology Commons, and the Science and Mathematics Education Commons

Recommended Citation
Franks, Robert Dee, "Effectiveness of training on algorithms versus notation for indirect addressing comprehension " (1992). Retrospective Theses and Dissertations. 9831.
https://lib.dr.iastate.edu/rtd/9831

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I
University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700  800/521-0600
Effectiveness of training on algorithms versus notation for indirect addressing comprehension

Franks, Robert Dee, Ph.D.
Iowa State University, 1992

Copyright ©1992 by Franks, Robert Dee. All rights reserved.
Effectiveness of training on algorithms versus notation for indirect addressing comprehension

by

Robert Dee Franks

A Dissertation Submitted to the Graduate Faculty in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY

Department: Professional Studies in Education
Major: Education (Research and Evaluation)

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Education Major

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1992

Copyright © Robert Dee Franks, 1992. All Rights Reserved.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ABSTRACT</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHAPTER I</strong> INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Need for the Study</td>
<td>11</td>
</tr>
<tr>
<td>Purpose of Study</td>
<td>11</td>
</tr>
<tr>
<td>Research Problem</td>
<td>12</td>
</tr>
<tr>
<td>Research Questions</td>
<td>12</td>
</tr>
<tr>
<td>Assumptions</td>
<td>13</td>
</tr>
<tr>
<td>Delimitations of the Study</td>
<td>14</td>
</tr>
<tr>
<td>Definition of Terms</td>
<td>15</td>
</tr>
<tr>
<td><strong>CHAPTER II</strong> LITERATURE REVIEW</td>
<td>18</td>
</tr>
<tr>
<td>Programming Instruction and Student Readiness</td>
<td>18</td>
</tr>
<tr>
<td>Initial Investigations</td>
<td>22</td>
</tr>
<tr>
<td>Research on Segments of Code and their Connection</td>
<td>25</td>
</tr>
<tr>
<td>Procedural Reasoning</td>
<td>30</td>
</tr>
<tr>
<td>Development of Procedural Reasoning</td>
<td>30</td>
</tr>
<tr>
<td>Acquisition of Mental Models or Knowledge</td>
<td>34</td>
</tr>
<tr>
<td>Schemata</td>
<td>31</td>
</tr>
<tr>
<td>Computer-based Simulation</td>
<td>34</td>
</tr>
<tr>
<td>Summary of the Research</td>
<td>35</td>
</tr>
<tr>
<td><strong>CHAPTER III</strong> METHODOLOGY</td>
<td>37</td>
</tr>
<tr>
<td>Subjects</td>
<td>37</td>
</tr>
<tr>
<td>Introductory Computer-based Materials Description</td>
<td>38</td>
</tr>
<tr>
<td>ALGORITHM</td>
<td>39</td>
</tr>
<tr>
<td>NOTATION</td>
<td>40</td>
</tr>
<tr>
<td>Material to be Learned</td>
<td>44</td>
</tr>
<tr>
<td>Dependent Variables of the Study</td>
<td>45</td>
</tr>
<tr>
<td>Instruments</td>
<td>47</td>
</tr>
<tr>
<td>Background questionnaire</td>
<td>47</td>
</tr>
<tr>
<td>Computer-based evaluation</td>
<td>48</td>
</tr>
<tr>
<td>Paper-and-Pencil evaluation</td>
<td>49</td>
</tr>
<tr>
<td>Research Hypotheses</td>
<td>52</td>
</tr>
<tr>
<td>Research Procedure</td>
<td>55</td>
</tr>
<tr>
<td>Method of Analysis</td>
<td>56</td>
</tr>
<tr>
<td><strong>CHAPTER IV</strong> RESULTS</td>
<td>58</td>
</tr>
<tr>
<td>Summary of Findings from the Computer-based Evaluation</td>
<td>59</td>
</tr>
<tr>
<td>Summary of the Findings from the Paper-and-pencil Evaluation</td>
<td>64</td>
</tr>
<tr>
<td>Testing the Hypotheses</td>
<td>69</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1  Computer-based Evaluation Descriptive Statistics for Problems Correct and Problems Response Time 60

Table 2  ANOVA Test for Group Differences on Computer-based Evaluation for Problems Correct 62

Table 3  Computer-based Evaluation ANOVA for Response Time 63

Table 4  Paper and Pencil Evaluation Descriptive Statistics 65

Table 5  \( \chi^2 \) Test with Yates Continuity Correction on Paper and Pencil Evaluation 67

Table 6  Paper and Pencil Evaluation ANOVA of Groups on Problems Favoring One Group 68
LIST OF FIGURES

Figure 1: First NOTATION Exercise Example . . . . . . . 41
Figure 2: Example Table used in NOTATION. . . . . . . 42
ABSTRACT

This study compared two units of instruction for overcoming difficulties of beginning programmers in understanding and implementing strategies of indirect addressing. One of the units emphasized the algorithms in which indirection was used, whereas the other unit emphasized indirect notation. Both instructional units were delivered by computer to two introduction to Pascal programming classes. Students in each class were randomly divided so that each student used one or the other of the two units.

These units were used prior to and were supplementary to three lectures covering indirect notation. The effectiveness of the units were determined by two posttests. One posttest, requiring students to select subscripts at several levels of indirection, was administered by computer. This test was very similar to the activities in the notation unit. The second posttest was a paper pencil activity requiring the students to complete or modify sorting algorithms in which indirection was used.

Because of the explorative nature of this study and the small sample size, the findings must be viewed as tentative. However, it would appear the notation of indirection by itself is not an important source of student problems in this area. In fact, there was little evidence to suggest that either unit
made a sizeable difference in the student's ability to deal with indirection within the context of programming. There was an indication extra study of algorithms encouraged students to attempt to solve more problems, but this finding may be a result of the experimental conditions and may not be generalizable.
Strengthening students' ability to analyze and solve problems is a major educational goal. In order to accomplish this goal, some educators have turned to computer science education (Papert, 1980). The problem solving skills required for computer programming have great appeal to researchers because of the skills' potential transfer to nonprogramming environments. The process of acquiring those skills may serve as the foundation for a general model for teaching problem solving.

Unfortunately, a reliable methodology for teaching students to use programming languages in order to solve problems has not been determined. Historically, a significant proportion of students who begin the study of programming have not been successful. Many of those who do complete the coursework are criticized for their inability to develop logical and concise software products. These facts have motivated some researchers, interested in the acquisition of cognitive skills, to investigate the teaching and learning of programming (Mayer, 1988; Pea & Kurland, 1983; Sheil, 1981b).

The methodology for introducing programming to students has closely paralleled that for presenting natural languages. With natural language the student first learns the grammatical or syntax structure of a sentence, then develops meaningful
(semantics) single stand alone sentences, and finally places sentences together to form a well-reasoned idea termed a paragraph.

Similar to natural language, the cognitive demands for learning programming fall into three parts: (a) knowledge of program language syntax; (b) knowledge of blocks of program code used for recurring tasks, also termed plans (Soloway, 1986); and (c) the ability to produce a set of instructions to achieve an ultimate goal, termed procedural reasoning (Sheil, 1981a). The level of knowledge a student must possess in order to successfully program is procedural reasoning knowledge.

Some researchers have suggested these parts form the sequence in which programming is learned and taught (Fay & Mayer, 1988; Linn, 1985). An example of such a sequence follows: a student knows how the WHILE statement (a syntactical element) in Pascal must be written; understands how to use the WHILE statement to terminate the processing of data; recalls from memory previous situation when the statement has been used; and applies that knowledge along with other plans in solving a problem such as finding the mean of an indeterminate number of values.

Other researchers advise the formation of procedural reasoning ability implied by the last step in the example sequence will not occur by teaching in this manner and that
the example sequence only reflects the first two cognitive
demands of syntax knowledge and blocks of code. They state
programming is too complex a task to be understood by a list-
of-facts approach (Sheil, 1981a). In other words, rote
learning of a list of plan sequences for solving specific
problems does not evolve into procedural understanding. A
strategy, specifically procedural, representing the machine's
behavior is required to correctly achieve problem solutions.
As an example of procedural reasoning that is more complex
than the list-of-facts approach, suppose a student knows how
to write a programmed solution to the problem of putting a
sequence of values in ascending order. The student also knows
how to compare two arrays to determine if they contain the
same elements. This student is presented with the problem of
determining if any two arbitrary words are anagrams, words
which have the same letters but in a different pattern. The
student who possesses an understanding of the concept of
sorting will recognize anagraming can be accomplished by
sorting and comparing arrays. With the list-of-facts
approach, it would be unlikely a student would connect the
sorting procedure with the anagraming problem since the terms
sorting or rearranging characters are not found in the problem
statement.

The resolution of the anagram problem requires the
letters to be listed in ascending order and compared. If all
letters are identical between the words then the words are anagrams. One domain consists of the combination of organizing a sequence of values in ascending order and of comparing sequences. The other domain is organizing two sequences of letters and comparing them. A near transfer of conceptual knowledge, or procedural reasoning, (Clark, Blake, & Knostman, 1989) has occurred between the similar domains.

The knowledge of syntax, which requires factual recall of given statements, is the least cognitively demanding of the three programming knowledge components and of the least interest to researchers of instructional methodology. Researchers' greater interest rests primarily in the other two programming knowledge elements, specifically, the acquisition of plans, and the development of procedural reasoning to transfer plans between programming environments.

A major problem in investigating the divisions of programming knowledge is severing the dependencies one part has upon the others. For instance research in writing (Scardamalia, Bereiter, & Goelman, 1982) implies that when there are great cognitive requirements of basic skills such as producing syntactically correct sentences, higher order skills such as story composition are weakly carried out. Fay and Mayer (1988) suggest removing constraints on lower level cognitive skills such as having the computer provide the programming language syntax, would allow more cognitive
resources for higher level skills such as problem solving. This suggestion implies cognitive connections of information transfer might be examined in greater detail if the syntax recall requirements of the student were somehow reduced.

One technique for diminishing the cognitive demands on the programming student attempting to overcome the hierarchy of programming knowledge is to use computer-based instructional simulations. Simulations which are used prior to formal instruction to prepare the student for future learning are called experiencing simulations (Thomas & Hooper, 1989). The design of the environment, choice of manipulations which can be made, and selection of activities determine the cognitive demands placed on the student. By incorporating features into the simulation which solve portions of a problem, the student can be exposed to activities dealing with different cognitive level skills mutually exclusive of the other levels. Development of these simulations requires a deep understanding of the cognitive needs of the students.

Studies performed by Hooper and Thomas (1990) have shown use of an experiencing simulation, involving procedural reasoning, influenced the student's choice of algorithms for solving problems. However, the student's performance on the syntax and use of blocks of code (plans) was unaffected. Thus, simulations may provide a way to guide students through areas of programming that traditionally (in the three part
sequence of programming knowledge) have been more difficult to grasp by providing activities which lie at the heart of the problem without encompassing the extraneous distractions.

Areas in which investigators have begun using experiencing simulation include computer memory operations, looping control constructs, procedures with parameter passing, and file processing (Thomas & Lewis, 1990). One aspect that each of these areas entails is expanding the concepts of variables, values and their relationship. In understanding computer memory operations, the student connects the idea that a variable is a symbolic representation for a memory location. The memory location can contain a value which can either come from input or assignment statements. Looping control constructs require the recognition of properties that the values possess. For instance, the values may be negative or may be in ascending order when taken as a sequence. Parameter passing generalizes the notion of symbolic representation of a memory location by allowing one location to have a host of symbolic aliases. With file processing, the student learns how values can be directly transferred between memory locations and devices connected to the machine.

None of these areas, however, specifically examine the nature of a variable’s values. It is the development of this understanding in which experiencing simulations may play a significant role. For instance, the solution to the problem
of finding the sum of a sequence of integers uses a variable to accumulate the integers as they are encountered. The nature of this accumulation variable is its possession of the actual data in some arithmetically-altered form. The numerical values of the data will change the contents of the accumulation variable. If the summation problem is modified to count the integers in the sequence, the solution will have a different type of variable. The variable will tally the sequence. The numerical values of the data will have no direct effect on the tally variable. It is the existence of an element of data, not its value, that modifies the tally variable. Thus values of input data directly effect some variables and indirectly effect other variables. Using experiencing simulations to create an environment in which students confront these subtleties before formal instruction may allow students to more easily assimilate this relationship.

This notion of indirect action or indirection runs throughout computer programming and has yet to be researched. The term indirection encompasses situations in which an object may contain some attribute which can be used to reference other objects. For example, a social security number is an indirect reference to a person or a license plate indirectly identifies a specific car and owner. A mailbox substitutes for residents of a dwelling so that the postal service does
not need to coordinate mail delivery with a resident being home, thereby improving efficiency of the service. The mailbox indirectly references the residents. Similarly, indirection in programming provides a more efficient means of manipulating large amounts of information by using the indirect reference as a key to the data. The keys can be processed while the data can remain unaffected.

In computer programming knowledge, indirection may be used at the syntactical level such as determining the scope of an identifier or retrieving fields within nested records. The sequence of indirect references, such as post office to mail carrier to mailbox to resident, is termed a chain. In an example of fields within nested records, the chain of indirection is static. For instance, given the following Pascal record:

```pascal
A : RECORD
  B, C : RECORD
    D : INTEGER;
    E : REAL;
    F : PACKED ARRAY [ 1 .. 5 ] OF CHAR
  END;
G : BOOLEAN;
```

the path to get to B’s second field will always be A.B.E. The record A will not change throughout the execution of the program. The record will always maintain the same form. Field E will always be the second field of that record. With
a static use of indirection, the object containing the indirect reference is constant.

In the latter two areas of programming knowledge, plans and procedural reasoning, the indirection is dynamic such as indirect addressing, parameter passing by reference, and implementing an abstract data structure. An object containing an indirect reference in these situations will be a variable which means the reference can change. Using a variable rather than a constant is an increase in the level of abstraction. It is in the dynamic nature of indirection in which student's programming problems arise.

As an example of dynamic indirection in procedural reasoning, suppose there exists an array \( Y \) with 25 values where each value ranges from 0 through 10. If the frequencies of each value for the range is desired (that is the number of ones, twos and so on), an array \( X \) could be used such that reference \( X[ Y[ I ] ] \) would map the \( I^{th} \) value of \( Y \) to the cell of \( X \) holding the frequency of that value. No longer would \( Y[ I ] \) be any specific value but would vary from 0 through 10.

The dynamic nature of accessing data is also forcefully demonstrated in dealing with pointers which are used for implementing data structures. The objects which pointers denote are typically located through the use of a unary operator and have no other name aside from the one given the object by pointer terminology (in Pascal for pointer \( P \), the
associated object is referred to by P^). A pointer can identify any properly typed object that has been dynamically created. Thus, a pointer is not bound to one specific object during its lifetime which is unlike static variables which have programmer assigned names. Students typically display great difficulty with pointers.

In developing programs which require the use of indirect access notation many students also experience considerable difficulty. Frequently, the problems which students exhibit appear in notation such as X[Y[I]] or P^.Link^.Info. In these cases many students will substitute an extra variable which serves as a placeholder such as Z := Y[I] so that the indirect reference is not nested, X[Z]. The execution of the notations is functionally the same but the resources required are different. There is a question, however, of whether this avoidance of notation reveals the problem of a deficient or immature mental image of indirection, is a symptom of failure to comprehend indirection or is a product of some other factors.

To use indirection because it exists is of no benefit. The meaningful use of indirection or indirect notation occurs when it is included in relatively complex algorithms such as sorting and searching. Unfortunately, beginning students may not have completely internalized the workings of such
algorithms. Student problems with indirection could be manifestations of problems with algorithm development.

Need for the Study

Indirection appears in many guises in computer programming. Many useful programming applications exhibit dynamic behavior which require indirection concepts. Unfortunately, some students go to great lengths to avoid dealing with different aspects of indirection. To promote the transition from student to expert programmer, more effective ways must be found to teach indirection.

Since indirection is foreign to beginning programmers, an experiencing simulation, which would provide a framework for learning the details of indirection, could be an effective learning aid. Before such a simulation can be developed, however, much more must be known about the factors impeding student learning. Initially, research is needed to identify major causes of student problems in this area.

Purpose of Study

The purpose of this study was to compare two supplementary units of instruction in an undergraduate beginning computer programming class. One unit was designed
to enhance student ability to handle the notation and the other to improve the grasp of the algorithms. The effects of the supplementary units of instruction on beginning programmers was analyzed.

Research Problem

The research problem for this investigation was to determine if the difficulties students encounter in indirect addressing was attributable to weakness in understanding algorithms, or to deficiency in handling the indirect array notation. Specifically, the problem was to ascertain if students exposed to two different focuses of supplementary instruction, notation and algorithm, differed on measures of comprehension.

Research Questions

One group of students was given training on indirect addressing notation which made use of a simulation, called NOTATION while a comparable group was given training and practice in using sorting algorithms with indirection, called ALGORITHM. Three fundamental research questions were addressed by this study.

1. Will the performance of the NOTATION group on
supplying subscripts to differing levels of indirect addressing be superior to that of the ALGORITHM group?

The results of this question was analyzed to assess the effectiveness of the notation training. Data for answering this question was obtained from a class of beginning programming students.

2. Will the performance of the ALGORITHM group on modifying sort routines which use indirect addressing be superior to that of the NOTATION group?

3. Will the performance of the NOTATION group on problems of indirect array manipulation which do not involve sort routines be superior to that of the ALGORITHM group?

The results of these latter two questions were compared to assess the relative effectiveness of the two methods and analyzed to identify basic student weaknesses in understanding indirection. Data for answering these questions were obtained from two classes of beginning programming students taught in two different semesters by two different instructors.

Assumptions

The assumptions of this study included the following:

1. The addition of this experiment to the course did
not in and of itself favor either group.

2. Although the experiment was performed over several semesters, the student population differences did not influence the experimental outcome.

3. The instructor differences over the semesters did not adversely effect the experimental outcome.

4. The treatments and duration of the experiment were sufficient to produce measurable results.

Delimitations of the Study

The delimitations of this study included the following:

1. Because the study was conducted at Central College the total number of participants in the classes used in the study was small which restricts the generalization of any findings of the study.

2. The students were volunteer participants and did not have tangible rewards which may have compromised their concentration during the study or willingness to put forth best efforts.

3. The material was extracurricular to the course requirements and the students were not examined over it by the instructor.

4. There was no control group that received
neither the ALGORITHM unit nor the NOTATION unit because of the potential small sample size.

5. The presenter of the indirection material was different from the instructors who taught the course.

Definition of Terms

Abstract data structure - a logical organization of data and a set of operations on that organization which is independent of implementation in a programming language.

Array - an abstract data structure of a logically connected collection of data which is all of the same type.

Dynamic chain - a sequence of activities which may change during the execution of a program.

Experiencing simulation - a computer-based simulation which is used prior to formal instruction on a topic to provide a framework, remove misconceptions, or otherwise facilitate the learning.

Indirection - the process of using value locations or identifiers as values themselves.
Looping construct - an algorithmic paradigm which causes specific instructions to be repeated in a determined manner.

Plan knowledge - the understanding of the semantics or meaning of a statement or small group of statements which solves a specific, limited task.

Pointer - an abstract data structure which contains the memory location of an object.

Procedural reasoning - "the process by which one determines the effect of a set of instructions or the set of instructions that will achieve a particular effect" (Sheil, 1981a).

Record - an abstract data structure of a logically connected collection of data which may be of different types.

Scope - the locations within a computer program in which attributes about an object can be obtained. For example, the places in a program the value of a variable be can accessed is the scope of the variable’s value.

Searching - the process of determining the existence and possible location of a datum.
Simulation - a computer program containing a manipulatable graphic model of selected aspects of a computer system. The program enables the student to change the model from a given state, through immediate states, to a specified goal state. The program accepts commands from the student, alters the state of the model, and then appropriate displays the new state (Thomas & Lewis, 1990).

Sorting - the process of putting data in some predetermined order.

Static chain - a sequence of activity which does not change during the execution of a program.

Variable - a logical name for a specific block of memory which can contain data.
CHAPTER II LITERATURE REVIEW

This chapter reviews the research literature from the areas of computer science education and psychology of learning which pertain to this study. The chapter has five sections. In the first section, student readiness for traditional programming instruction is discussed. The next three sections comprise a summary of research on the acquisition of each aspect of programming knowledge in the order it has been traditionally presented; syntax, blocks of code, and procedural reasoning. These sections focus on the syntactical structure of programming languages as it affects learning and understanding, the learning of segments of code and modification of those segments, and ways of facilitating student acquisition of procedural reasoning. The final section contains a brief summary of the chapter.

Programming Instruction and Student Readiness

Programming instruction has traditionally followed the model of natural language learning with all its ambiguities. This is a hierarchical model in which the syntax or grammar is first presented followed by the derivation of the semantics or meaning from syntactical expressions. Fay and Mayer (1988) described this model in terms of the learners' cognitive tasks.
The first task is a code recognition/production task in which the student has to recognize and generate legal programming code. This task requires knowledge of the programming language syntax. The second task is a comprehension task. The student has to provide the outcome that would be produced for a given set of instructions. The student must have a knowledge of language semantics as well as skills from the first task. The last task, dependent on the previous two, is the creation of a set of instructions to solve specific problems. This task requires the student to transfer knowledge acquired in the programming environment to a new problem solving domain. Clark, Blake and Knostman (1989) termed this "far transfer." Far transfer occurs when information in a familiar source schema located in one domain is connected to information in a different and less familiar target domain.

Because the task of creating sets of instructions results in computer programs, it is the ultimate goal of programming instruction, the object of internal and external criticism and the focus of much research. However, improvement of programming instruction requires a much broader view, including an analysis of the previous experiences and preexisting knowledge of incoming students. The preexisting knowledge that a student possesses when entering any learning situation is of notable consequence. Gardner (1991) asserted
by the time the formal schooling age is reached, children have already developed specific intellectual strengths and styles. Because of these strengths and skills, children may have difficulty in school for several reasons. First, the taught material may be alien to many students because the context in which the material originated has lost its vitality. Second, some of the notation and conceptual frameworks are not easily conquered by students whose intellectual strengths are aligned differently. Finally, the required form of knowledge may conflict with the child's previously developed and robust forms of knowledge. All of these difficulties beset beginning programmers.

Furthermore, Resnick (1983) stated that all students come to their first class in any of the sciences with extensive "naive" theories to explain natural phenomena. After instruction with new scientifically accepted concepts, students will return to those "naive" theories when asked to solve any problem which deviates from that specific instruction.

When learning programming, according to Sheil (1981a), any preexisting theory is "naive." A student may analogize programming to giving directions. Such an example leads to the anthropomorphism of the machine which is contrary to reality. The student places greater emphasis on the machine's understanding of what is meant by the directions than on the
literal content of the directions. In actuality the directions or instructions contain all meaning, and the machine makes no judgments. Therefore, Sheil claimed this example of preexisting skills does more harm than good.

Contrary to Sheil's blanket assertion, pieces of useful preprogramming knowledge can be found in natural language processing. Bonar and Soloway (1985) noted natural languages contain the concepts of looping, selection and sequencing which are essential in programming. Unfortunately, the surface similarities between natural languages and programming languages can cause difficulties for students when words from natural language are present in the programming language.

Thus, because of the unique nature of programming, its complexity, and the characteristics of beginning students, severe learning difficulties are inevitable. Many of these difficulties reappear as new and unique topics are encountered throughout the study of the discipline. The realization of these difficulties, or at least their symptoms, has encouraged and continues to encourage research on the teaching of programming.

While the primary purpose of a computer program is to instruct the machine as to what operations to perform, it also serves as a tool for conveying concepts and algorithms. Because this tool may be repeatedly refined over a period of years by a multitude of programmers, it must be expressed in a
comprehensible manner. This need for an understanding by a wider audience brought about the evolution of program representation from binary to forms similar to natural language or mathematical notation. Unfortunately, there was little guidance in how best to make a program comprehensible.

The following is a chronological perspective of attempts at making programs more easily understood by those not involved in the creation of the programs. Attention then focuses on the development of research into the cognitive processes of learning to program.

Initial Investigations

The first significant criticism of programming style was made by Dijkstra (1968). He claimed that programmers’ excessive use of the GOTO statement made program comprehension more difficult. Dijkstra’s concern focused on the idea that a program was a static document representing a dynamic activity. He asserted that the GOTO statement obscured the programmer’s comprehension of the transition from static objects to dynamic objects. Although Dijkstra did not offer external evidence to support his assertion, his criticism motivated several studies of the role of notation in programmer performance.

Sime, Green and Guest (1973) examined the notation of conditional statements using nonprogrammers to select proper
cooking procedures given a set of initial objects. The study compared the nested **IF THEN ELSE** notation with the conditional **IF GOTO** notation. The study found the structured **IF THEN ELSE** notation to be superior. Studies by Weissmann (1974) and Lucas and Kaplan (1976) scrutinized the structured looping constructs from a block structured programming language, PL/1. The study sought to determine the difference in maintainability of code using structured loops with code using **GOTO** statements. The study determined programmers felt more confident in the correctness of the structured solution but found no difference in programmer performance in comprehension and debugging both groups of code. This work was criticized for lacking analysis on the way programmers approach the programming task either in a structured or unstructured way. These aforementioned studies did not take into account the difference between experienced programmers and novices.

Experienced programmers have a greater grasp of the dynamics of a program which is a static object. Because of the input, a program may behave differently each time it is executed. Techniques used to improve a programmer's understanding of a program's dynamic behavior such as flowcharting, indenting, and commenting were observed in several studies. Shneiderman, Mayer, McKay, and Heller (1977) analyzed the once popular flowcharting to determine which types of programming tasks were enhanced by the use of
flowcharting. They failed to find any reliable advantage of using flowcharts thereby placing the burden of empirically validating the positive claims of flowcharting on its advocates. Shneiderman and McKay (reported in Shneiderman, 1980) also found claims for indenting to be overrated. When programmers were given the task of finding errors in Pascal programs, there was no significant difference between those programmers using indented programs and those with identical unindented programs.

Shepard, Curtis, Milliman, and Love (1979) found that program description level comments and statement description level comments had no reliable effect on small FORTRAN program modification in terms of time or accuracy. Comments appeared useful only when the programmer did not know the fact or could not easily deduce it from the code.

These studies contributed little to computer science education, having much less impact than Dijkstra's critical remarks. Part of the impotency can be attributed to unsophisticated experimental method but, as Sheil (1981b) observed, the major factor was the researchers' failure to understand the complexity of programming skill. The research investigated the least challenging aspect of programming and consequently the least difficult to learn. Even if much had been discovered and incorporated into the instruction, it is unlikely the discoveries would have made a major difference in
the programming deficiencies Dijkstra was criticizing. However, failure of these efforts discouraged continued study of single statement comprehension and led to research on selecting and adapting blocks of program statements.

Research on Segments of Code and their Connection

Blocks of program statements to perform a particular function are one example of what Hiebart and Lefevre (1986) defined to be conceptual knowledge. It is the view of these researchers that conceptual knowledge is a web of knowledge, rich in relationships, in which the linking relationships are as prominent as the discrete pieces of information. They believe conceptual knowledge is developed by the forging of relationships between existing pieces of information and new information being presented. The establishment of relationships can be accomplished either on a primary level in which all the connecting information is at the same level of abstraction as the relationship or on a reflective level in which the relationship is at a higher level of abstraction. Investigations of connections made at the same level of abstraction focused on blocks of code.

Over several years with related studies involving segments of code (Soloway, Bonar, & Ehrlich, 1983; Bonar & Soloway, 1985; Letovsky & Soloway 1986; Soloway, 1986),
evidence was collected to support the position that teaching syntax and semantics of the statements in a language in and of itself was not enough to make students good programmers. This research thrust was initiated with attempts to gain empirical evidence as to what novices know about programming (Soloway, Ehrlich, Bonar, and Greenspan, 1981). It was motivated by a desire to create automated teaching systems which could understand student difficulties and provide appropriate guidance. The research confirmed Shell's position on the complex nature of programming and the use of programming languages.

For instance, the Soloway et al. (1983) study showed when students were given programming problems where the most efficient solution involved one looping construct, **REPEAT UNTIL**, most students attempted convoluted solutions by using a different looping construct, **WHILE**. Soloway et al. speculated that students using the **WHILE** construct were more "comfortable" or that the construct had a "greater cognitive fit" with their own models.

In terms of Resnick's assertion about "naive" theories, the students had examined the given problem which was different from others they had done but did require looping with which they were familiar. They connected the fact that the problem necessitated a loop with the **WHILE** loop construct. Finally, they added other pieces of code to the **WHILE** loop in
an attempt to solve the problem. The result was that the solution was more complex than needed or that the solution was incomplete.

Later studies by Soloway et al. (Bonar & Soloway, 1985; Letovsky & Soloway, 1986; Soloway, 1986) showed the importance of students being able to form blocks of statements to accomplish specific tasks. Successful problem solutions required the novice to master a set of these plans which would be used in conjunction with each other. Thus students needed to understand the relationships between a set of statements in order to achieve the desired effects.

Bonar and Soloway (1985) observed that a student’s ability to write correct programs was fragmentary. Some portions of a program were correct while other portions were incomplete. Soloway (1986) noted the incompleteness was due to lack of comprehension in putting separate programming plans together. He described four plan placement strategies of abutment, nesting, merging, and tailoring as required for achieving a correct procedural solution to a problem. Soloway describes abutment as two plans placed sequentially together whereas nesting placed one plan within another. With merging, the student must have two or more plans blended together in one unit. A specific problem may have required a plan to be tailored to a different set of conditions than for which the plan was originally designed.
This work rejuvenated suggestions that programming languages should include complex plans as primitive constructs. But, as Sheil (1981a) had noted, defining a programming language to include such composite plans as a primitive construct would not alleviate the need for procedural reasoning abilities on behalf of the student. Any such composite can be encompassed into a larger composite by changing the specifications of the given programming problem. To achieve a solution students must develop the capability to make associations between knowledge elements and to assimilate knowledge units.

Making connections between pieces of information is referred to as transfer. Clark, Blake and Knostman (1989) described "near transfer" as the association formed within a domain where knowledge is formed hierarchically whereas the previously mentioned "far transfer" is between different domains. Near transfer is promoted by example and by analogy. Examples and their related rules are one of the most important components of knowledge hierarchies as students learn and use concepts. Students transfer these rules when they make the decision that a new experience is an instance of a more general principle. In some sense, these connections are vertical in that they exist within a hierarchy where examples are subordinate to rules and rules are superordinate to
examples. Example level connections primarily support learning within a specific domain but not between domains.

Learning a set of rules is not enough to become a proficient programmer. Sheil (1981a) argues list-of-facts based connections fail in providing understanding of a complex programmed device. He suggests a procedural theory is required to represent the knowledge about the device's behavior, that is, the blocks of code must be envisioned in such a way as to create a procedure or algorithm which can then be used to describe the device outcomes.

The student evolves into the expert as procedural reasoning ability improves. This structuring of information for efficiency and comprehension is similar in nature to how expert chess players recall chess piece positions on a board. Chase and Simon (1973) found that chess experts could remember board positions significantly better than novices when the positions had some structure, positions which could be encountered in a game. When the pieces were randomly assigned on the board, there was no difference between expert and novice in recall. An understanding of the transition from novice to expert or from plans knowledge to procedural reasoning knowledge can be viewed in the work which follows.
Development of Procedural Reasoning

In a review of work on knowledge acquisition, Vosniadou and Brewer (1987) state that knowledge is structured in the form of schemata which can be modified by new experience. They characterize accretion, tuning, and restructuring as types of change that occur with the gaining of new knowledge.

Accretion refers to the change resulting from the gradual accumulation of factual information within the existing schema. Tuning describes the evolutionary modifications in the classifications used for interpreting information or Clark, Blake and Knostman's near transfer by example. Restructuring creates new structures, either from the reinterpretation of old information or taking new information into account, the analogy connection. Integrating new information to preexisting knowledge is termed weak restructuring. If the schema is incorrect, insufficiently generalized, or cannot be identified, then radical restructuring is required. Because assumptions about structures become inconsistent, they need to be radically transformed.

Student procedural reasoning knowledge is often fragmentary, requiring radical restructuring of knowledge
schemata. Resnick (1983) asserted that for understanding to develop the student must undergo a long and repetitious process. She suggested that because naive theories are inevitable, students should be forced to place their theories against the ones they are being taught. The students will then learn to deal with conflict in theories in a similar way to scientists.

In a similar vein, Gardner (1991) states all understanding is partial and subject to revision. He declares it is far more important for a student to understand the processes whereby misconceptions are removed than for an arrival at a "correct view." The role of formal education should provide students with exposure to new ways of conceptualizing familiar and unfamiliar entities. By focusing on the qualitative aspects of problem situations students will have time to acquire the types of analytical and notational skills needed. By confronting naive mental models directly, students are forced to compare their models with those which they are asked to learn.

Acquisition of Mental Models or Knowledge Schemata

An initial investigation on acquiring correct mental models was the study performed by Mayer (1981) which applied Ausubel's (1968) concept of advanced organizers. Mayer
provided one group of programming novices a concrete model of a computer and taught each statement in the language in relation to the actions the model would perform on that statement. Another group was allowed to create its own mental image of what occurred during statement execution. The results indicated that by providing a model, assimilation of new statements was improved and novices could encode information more clearly.

However, in a followup study, Bayman and Mayer (1983) found those who were defined as high ability novices were not significantly helped by the model and on some tests were in fact hindered by the model. This fact indicated that some novices had already internalized their own models for what they were experiencing and that the research model was confounding their performance.

Du Boulay, O’Shea, and Monk (1981) stated that novices begin programming with little idea as to the properties of the notational machine implied by the programming language. They hypothesized simplicity and visibility were the guidelines when creating a model of a computing machine for a specific programming language. It was suggested that the model machine should consist of a small number of components that interact in ways that are easily understood thereby assisting the novice in learning the language.
In a review of his previous work Mayer (1989) examined the usefulness of providing conceptual models to assist novices in learning scientific concepts. A conceptual model was defined as words or diagrams designed to help novices build mental models. The conceptual model highlighted major components in the concept along with any internal or external interaction of those components. The research examined met four criteria: learners were novices; the to-be-learned material was explanatory rather than descriptive or narrative; conceptual models were used as aids to instruction; the dependent measures were conceptual information recall, retention of material in verbatim format, and creative problem-solving transfer performance.

Given these conditions Mayer stated that novices using a model would recall more conceptual information than those without a model as the model would guide novices' selection of material for learning. Model-using novices would also be less likely to retain verbatim because they had reorganized the material and integrated it into long-term memory. Finally, model-using novices would be more likely to generate creative solutions to transfer problems because they would be mentally manipulating a model. Because this analysis involved only those novices who would be helped by a model, the highly skilled learners from Mayer's previous studies would not be
included. The results of the review confirmed Mayer's statements.

When scrutinized, Mayer's studies show the confounding nature of the naive theories that students possess. In order to evaluate the confrontation of naive views and desired views, measures of rules and examples should not be at the comprehension level of programming techniques. Only measures of near transfer will provide insight to the degree of acquisition of procedural reasoning.

Computer-based Simulation

One way to study this near transfer or procedural reasoning is to circumvent the traditional hierarchy of learning programming. Thomas and Hooper (1989) proposed one such approach through the use of a computer-based instructional tool termed simulation. They categorized simulation according to the role simulations played in formal instruction. "Experiencing simulations," as defined by Thomas and Hooper, provide an environment for introducing the user to a concept before the user has had a formal presentation of that concept.

In a study performed by Hooper and Thomas (1990), it was found that an experiencing simulation, modeling computer memory operations, influenced novices' selection of algorithms.
throughout the programming course. Such a transformation would be considered near transfer. However when it came to factual recall skills in dealing with statement syntax the simulation made no significant difference.

The experiencing simulation provided a dynamic model for students to learn the details of computer memory operations whereas Mayer’s models were static. The dynamics model may have gone farther into facilitating the transfer process. By taking advantage of the computer’s capabilities, novices were able to explore the memory operation model in ways not possible with a graphic static form. In an overview of the use of experiencing simulation with tasks which are traditionally difficult for programming novices; memory operations, looping structures, subprograms, and file operations, Thomas and Lewis (1990) stated that simulation also promoted the near transfer of procedural reasoning in these areas.

Summary of the Research

The early research into programming methodology centered on the teaching and use of specific instructions and program form. These studies made little contribution to the understanding of program learning because they failed to account for the complexity of programming skill. Subsequent
research analyzed the more difficult tasks of student use of blocks of code. However, both the early studies and the subsequent ones focused on ways to transmit knowledge to students.

As the limitations of these approaches were observed, researchers turned from seeking ways of transmitting knowledge to students to ways of facilitating student acquisition of knowledge. Researchers also turned from a simple view of programs to a view incorporating procedural reasoning. Mayer documented the importance of conceptual models and Hooper and Thomas demonstrated the utility of dynamic models. Before these models can be a major force, however, much research into the role they play and their structure must be completed.
CHAPTER III METHODOLOGY

This study was performed to compare two types of supplementary instruction, one focused on algorithm development with indirection and the other focused on development of indirect array notation using a simulation. The performance of the two groups was assessed on two posttests. One posttest required the student to provide subscripts for indirect addressing notation containing multiple levels of nesting. Assessing acquisition of skill in using indirect notation was the role of this test. In order to assess procedural reasoning, another posttest was given. In this test, the student was required to solve problems using indirect addressing.

Subjects

The subjects of this study were enrolled in Computer Programming 1 taught in the Fall 1990 and 1991 terms at Central College. Central College is a private undergraduate liberal arts institution with a student body of approximately 1500 which has an mean ACT of 24. Ninety percent of the students come from within one hundred miles of the campus. The Computer Programming 1 course does not satisfy any general education requirements. Students enrolled in the class have
some interest in computing. The researcher was not the instructor for the classes.

Participants were initially given a consent form and a background questionnaire (Appendix A). Thirty-two students from the Fall 1990 term and 16 students from the Fall 1991 term volunteered to take part in the study. Of the volunteers, 38% were female, 79% had at least one semester of Basic and/or Pascal programming in high school, 71% were enrolled in or had at least one term of Calculus, and 54% were classified as freshmen, 33% were sophomores, 10% were juniors and 2% were seniors. Two students for the Fall 1990 term and one student from the Fall 1991 term failed to take the two posttests.

Introductory Computer-based Materials Description

Two computer programs were developed to provide experience with the levels of the independent variable, algorithm manipulation and notation handling. These programs, entitled here as ALGORITHM and NOTATION (Appendix B), were written in the Digital Authoring Language and were made available to the students through the Central College MicroVax 3400 computer. These programs were first presented to a group of computer science majors as a trial run to eliminate anomalies in the material. The algorithms used may be found
in various programming textbooks such as one by Sedgewick (1988).

**ALGORITHM**

The ALGORITHM program provided the student with a textual overview of material which was presented in the class lecture. By going through the ALGORITHM activities, the student was given familiar searching and sorting algorithms. Those algorithms were presented with a new prospective which progressed into a rationale for using indirect addressing with arrays. In using this program, the student should also have acquired some intuition as to the application of indirection to sorting and searching algorithms.

The ALGORITHM program presented the user with two arrays which contained salaries and social security numbers respectively. An algorithm to find the highest salary was displayed which used a variable to hold the actual highest value. The program text emphasized that while the largest value was found, the location within the array of that value was not known. The program pointed out the importance of the value's location because the social security number associated with the salary might also be of interest. The social security number cannot be found without knowledge of the salary's location.
An alternative approach to finding the highest salary was shown which used a variable to hold the location of the highest salary instead of the salary's value. The ease in retrieving the corresponding social security number was stated and the concept of indirection using this example was explained. Actual values for salaries and social security numbers were listed along with their corresponding subscript. A location column was produced which ranked the data in descending order according to salary. The explanation of indirection was expanded using the example.

The algorithm for a selection sort to list the salaries in descending order was written at the top of the screen. Just below that algorithm, the program placed a modified algorithm using indirection. Using the data example, the modified algorithm was executed one step at a time to show how the location column received its values. A summary of the concepts presented was the last portion of the program. The only interaction the user had with the program was to indicate when to proceed to the next step in the program.

NOTATION

The purpose of the NOTATION program was to introduce the student to the concept of indirection through the use of array subscripting notation. By completing the program the student
was capable of providing the correct subscript for array accesses which use from zero to two levels of nesting of array references as subscripts. The student should then have a frame of reference to assist in solving problems which have such notation.

The NOTATION program began with exercises to familiarize the student with the ordering of data. The program presented the student with data under three columns; name, age and person (Figure 1). Starting with the letter "a", the person column delineated the name and age data. The student was asked to make two searches: one for the third youngest entry and another for the entry which would be placed second alphabetically. The program next asked the student to provide two rankings using the letter under the person column as the indicator of the order the entries should be placed: youngest to oldest (c,a,d,e,b) and name in reverse alphabetical order (a,e,c,b,d). The student was allowed three attempts to select the correct ranking after which the proper ordering was presented.

<table>
<thead>
<tr>
<th>Person</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>terri</td>
<td>38</td>
</tr>
<tr>
<td>b</td>
<td>ernst</td>
<td>82</td>
</tr>
<tr>
<td>c</td>
<td>jewel</td>
<td>14</td>
</tr>
<tr>
<td>d</td>
<td>akbar</td>
<td>52</td>
</tr>
<tr>
<td>e</td>
<td>sarah</td>
<td>62</td>
</tr>
</tbody>
</table>

Figure 1 First NOTATION Exercise Example
The program reversed the direction of questioning to begin to incorporate array notation into the demonstration. The same columns were shown with different values and a ranking was given. The array notation using the ranking was explained. The program stepped through each entry in the ranking and highlighted the corresponding entry in the data. The student was prompted to supply a subscript which caused both the rank entry and the data entry to be highlighted. This process was repeated for another ranking. At the completion of this portion of the program, the student was requested to supply data which made the ranking correct. The student was told whether data that had been given was in the proper order.

After an explanation of the concept of indirection using an example consisting of an array of ranks and an array of data, the exercise portion of the program displayed a five row by four column table as shown in Figure 2.

<table>
<thead>
<tr>
<th>Row</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**Figure 2** Example Table used in NOTATION
One of the entries under column W was highlighted. The array notation of \( W[\ ] \) was displayed. The student was asked to supply the subscript which would provide access to the highlighted element using the displayed notation. If the student entered the wrong subscript, the element that wrong subscript accessed was highlighted. The program stayed at this level of notation until the student had correctly supplied at least five subscripts.

The nesting level of the notation was increased by one to become \( W[\ ?[\ ]\ ] \) where "?" may be a column of either "X", "Y" or "Z." The procedure was repeated with the only difference being the notation. Once five correct answers were furnished, the notation was changed to \( W[\ ?[\ ?[\ ]\ ]\ ] \) where a column can appear only once. The last exercise had the notation of \( ?[\ ?[\ ?[\ ]\ ]\ ] \) where X, Y and Z all appear once.

At the completion of the notational exercise the student was shown the score for each level, the number of problems correct and the number of problems attempted. The student was given the opportunity to go back to any level of this exercise and repeat the activity. Lastly, a textual summarization of indirect array notation was shown to the student.
Material to be Learned

Each student was given a five page handout (Appendix C) which covered the concept of indirection with arrays. This material was presented during the initial class meeting when volunteers were solicited and background material was completed. The students were asked to read the document before the lectures were given. The lectures restated the contents of the handout. The lectures took place over two consecutive 50 minute class periods after the computer-based materials were completed.

The handout detailed a natural progression of algorithm development which embodied the use of indirection with arrays. First, the idea of using data as subscripts appeared as a problem of tallying quiz scores in a graph. The indirect array notation emerged when the data was stored in an array.

The handout next introduced an initially unrelated problem of anagrams, words which contain the same letters but in a different order. It demonstrated that the anagram problem was an analogous form of the graph problem in which indirection could be used. The difference was that anagrams dealt with character data and the graph problem had numeric data.

From that point the material progressed similarly to the ALGORITHM program. Two arrays containing salaries and social
security numbers were given. It was emphasized indirection needed to be included into the algorithm of finding the largest salary so that the associated social security number could be retrieved.

The need for an array of indirect references was introduced based on an example of a parking garage recording car license plates. The license plate example explored the integrating of indirection into both selection and bubble sorts. Both of the sorts were covered in class previous to the commencement of this study. The indirect array notation was embedded in the algorithms. The algorithms were traced through on the blackboard in class to show how the indirect array received its values. Finally, a problem based on the example was stated and then solved using indirection.

Dependent Variables of the Study

The independent variable of this study was supplementary training which had two levels, NOTATION and ALGORITHM. Seven dependent variables were used to measure the effect of the supplementary training units on the students.

The first dependent variable, dv la, which asked the student to supply the correct numeric subscript for an indirect array notation to access a given element in a table, was used to determine group differences in notational
comprehension. This variable dealt with factual recall of planning programming knowledge.

The second dependent variable, dv 1b, measured the group's ability to transfer notation understanding from a domain consisting of numeric subscripts to a domain consisting of non-ordinal symbolic subscripts. The students were asked to supply the correct symbolic subscript for an indirect array notation to access a given element in a table.

The remaining dependent variables, dv 2 through dv 7, dealt with near transfer or procedural reasoning, in an attempt to gain insight into the interplay of notation and algorithm comprehension. The sort routines from dv 2 through dv 4 would appear the same with or without indirection, the only exception being the comparison of elements. Dependent variable 2 (dv 2) checked for correct use of indirect array notation when attempting to reverse the comparison direction of a sort routine. Dependent variable 3 (dv 3) examined the correct use of indirect array notation when attempting to add an efficient termination condition of a sort routine. Dependent variable 4 (dv 4) inspected the correct use of indirect array notation when attempting to reverse the element replacement location of a sort routine. It was anticipated these dependent variables would provide insight on whether the algorithm confounded the proper application of indirection.
The dependent variables dv 5 through dv 7 attempted to determine what the student could infer about the original data by providing an indirect array consisting of the subscripts of the original data array. Dependent variable 5 (dv 5) measured the correct manipulation of the indirect array to provide an inverse array. Dependent variable 6 (dv 6) examined the correct identification of the logical distance between two elements in the original data when supplied with the elements of the indirect array. Dependent variable 7 (dv 7) assessed the correct discernment of the reason different sort routines may have different element orderings in the indirect arrays.

Instruments

Three types of evaluative instruments were used in the study. A background questionnaire, a computer-based evaluation of notation comprehension, and a paper-and-pencil evaluation of near transfer were administered.

Background questionnaire

A sequence of questions about the background of each student was given. Studies of prerequisite skills for programming success of the computer novice have been unsuccessful in determining a specific set of abilities
Therefore, questions pertaining to previous programming and mathematics courses, biographical information, current grade point average, and class rank were for verifying the equivalence of the groups.

Computer-based evaluation

The evaluation program (Appendix D) was similar to the last exercise of the NOTATION program which involved indirect array notation. This evaluation was used to measure the dependent variables of correctly supplying numeric (dv la) and character subscripts (dv lb) and the time required to so. The range of possible scores was 0 (all problems incorrect) to 16 (all problems correct) and the response time was measured in seconds.

The program provided a brief introduction to the indirect array access activity for the benefit of the ALGORITHM students. The student was given a five row by four column table with one of the entries highlighted and a notation below it. The notation and the highlighted entry were from a specified sequence. The students were told that their responses were being recorded. A total of 16 of the problems (four problems each for four levels) were presented. The correct answer, the given answer and the response time were
recorded for the last problem of each level. The student was not told if the response was correct. The student was given a point for each problem correct. Only the score for each level was displayed at the end of the problem set. In order to determine if the intrinsic ordering of the numeric subscripts was of importance, the activity was repeated using nonnumeric, nonletter characters. The same data were recorded for this activity as well. A student was therefore given 32 indirect array notation problems to solve.

**Paper-and-Pencil evaluation**

The purpose of this evaluation (Appendix E) was to determine if transfer of procedural reasoning had taken place. Recall that previous work with computer-based instructional simulations only showed significant effects in relation to transfer (Hooper and Thomas 1990). The problems in this evaluation were created and three experts independent of this study determined whether the problems favored either the ALGORITHM students who had more exposure to algorithms or the NOTATION students who had greater emphasis on notation. The criteria used to measure the responses to these problems can be found in Appendix F.

The first four problems required the student to modify the sorts, bubble and selection, and include indirect array
notation. These problems served as a measure for the dependent variables reversing direction of a sort routine (dv 2), adding an efficient terminator to a sort routine (dv 3), and reversing the element replacement location of a sort routine (dv 4). There were three measures for each of the problems; correct or not correct, attempted or not attempted, and a score. The first two measures are boolean, true or false, and the last measure is a whole number. The score for each of the first three problems was 0 or 1. Problem 1 asked the student to reverse the comparison direction of the sort given the bubble sort algorithm containing indirection. Problem 2 asked the student to add an efficient terminator for the bubble sort. Problem 3 asked the student to reverse the replacement of element in a sort given the selection sort algorithm containing indirection. Problem 4 asked the student to remove the indirection from the selection sort. The score for this problem was 0, 1 or 2 as there were two locations to remove the indirection. These problems were posed because the behavior of the sorts remain unchanged with indirection. Comprehension of how the sort routines function would allow the looping constructs to be modified and allow indirection to be easily integrated. The expectation would be that the ALGORITHM students would be more successful with these problems and have a greater likelihood of success in solving these problems.
The fifth problem asked the student to correctly identify the distance between the first and last elements in an array given only the indirect array. This problem did not involve the use of a sort algorithm. The dependent variable associated with this problem was the determination of the original data location given only the indirect array (dv 6). The student was given the values of an indirect array and was asked physically how far apart the logically first and last values were in the data array. This problem was somewhat similar to some of the activities found in NOTATION in which mapping took place from the given notation to an array containing data. The student needed to map the first and last values of the indirect array into the data array and count the number of elements that were in between. The score for this problem was 0, 1 or 2. It was hypothesized the NOTATION students would do better with this problem.

The sixth problem was another variation of one of the NOTATION activities. Given two indirect arrays, one containing values and the other empty, the student was asked to fill in the empty indirect array so that reversing the nesting of array references would yield equivalent values, 

\[ A[B[*]] = B[A[*]] \]

The dependent variable for problem six was to supply the inverse for a given indirect array (dv 5). The score for this problem was 0, 1 or 2. Again, the NOTATION students would be expected to perform better on this problem.
In order to measure the dependent variable, different orderings within a sort routine (dv 7), the seventh problem required the student to identify a reason why bubble and selection sort routines might have a different indirect array. The problem stated that two given indirect arrays, one used with bubble sort and one used with selection sort, were different even though the data arrays were identical. Even though indirect array values were given, the problem required the student to understand the difference between the two sorts when data values are not unique. In this problem, the indirect array reference merely showed the workings of the sort routines and was not central to the solution. Therefore, it would be assumed the ALGORITHM students would have a better grasp of the concepts required to answer this problem. The score for this problem was 0 or 1.

Each of the problems of the paper-and-pencil evaluation were attempts to determine if near transfer or procedural reasoning had taken place. The problems were limited to seven because of time constraints.

Research Hypotheses

The ten following hypotheses expand upon the three fundamental research questions. The first four hypotheses are an enlargement of the first research question which involved
supplying subscripts in indirect array notation. These hypotheses pertain to the assessment of proficiency in using addressing with zero, one or two levels of indirection:

1. The NOTATION group will perform significantly better than the ALGORITHM group in correctly supplying a numeric subscript (dv 1a).

2. The NOTATION group will perform significantly better than the ALGORITHM group in the time required to supply numeric subscripts (dv 1a).

3. The NOTATION group will perform significantly better than the ALGORITHM group in correctly supplying a character subscript (dv 1b).

4. The NOTATION group will perform significantly better than the ALGORITHM group in the time required to supply character subscripts (dv 1b).

The next hypotheses concern the assessment of near transfer. Hypotheses five, six, seven, and ten relate to research question two which measured student performance on modifying sort routines that incorporate indirect array notation. Hypotheses eight and nine relate to research question three.
which involved student performance on manipulation of indirect array notation not entailing sort routines.

5. The ALGORITHM group will perform significantly better than the NOTATION group in correctly reversing the comparison direction of a bubble sort (dv 2).

6. The ALGORITHM group will perform significantly better than the NOTATION group in correctly adding an efficient terminator for the bubble sort (dv 3).

7. The ALGORITHM group will perform significantly better than the NOTATION group in correctly reversing the replacement location of a selection sort with and without indirection (dv 4).

8. The NOTATION group will perform significantly better than the ALGORITHM group in correctly identifying the distance between the first and last elements in an array given only the ordered indirect array (dv 5).

9. The NOTATION group will perform significantly better than the ALGORITHM group in correctly supplying the inverse of a given indirect array (dv 6).
10. The ALGORITHM group will perform significantly better than the NOTATION group in correctly identifying why the indirect arrays from bubble and selection sort are different (dv 7).

Research Procedure

The activities of the study took place over the period of one week. The research design was a posttest-only control group design. At the completion of the chapter on one dimensional arrays from the textbook (Leestma & Nyhoff, 1990), the students were asked to participate in the study. The students were asked to sign a consent form, fill out the background questionnaire, read the lecture handout (Appendix C), and sign up for a time to perform the computer-based learning activity.

Later, on the day in which the students had a regularly scheduled examination over the in-class material, including one dimensional arrays, the students received the supplemental training activity. The students were randomly assigned to either the ALGORITHM program group or the NOTATION program group.

The two class periods, after the examination and training, were devoted to the lecture material on indirection. The material presented was the information which was written
in the handout. The only material the students had seen involving indirection was the handout presented by the study.

The class instructor resumed traditional instruction in the following class period. After a weekend passed from the time of the study lectures, the students returned to the lab to perform the computer-based evaluation, and the paper-and-pencil evaluation.

Method of Analysis

To determine the experimental effects on the computer-based evaluation for the Fall 1991 class, an analysis of variance test was used for each of the four levels for both numeric and character subscripts. The independent variable was the supplementary training group. The computer-based evaluation provided data for hypotheses 1 through 4 (dv la and dv lb). The number of problems correct using numeric subscripts provided data for each of the four levels (no indirection, one-level of indirection, two-level accessing the left column, two-levels accessing the remaining columns) of hypothesis 1. The number of problems correct using character subscripts provided data for each of the four levels of hypothesis 3.

The analysis of variance was performed on the response times for each of the four levels for both types of subscript.
where the independent variable was the supplementary training group. The response time, measured in seconds, for problems using numeric subscripts, provided data for each of the four levels of hypothesis 2. The response time, measured in seconds, for problems using symbolic subscripts, provided data for each of the four levels of hypothesis 4.

To determine the experimental effects on the paper-and-pencil evaluation, a chi-square test for independence with Yates continuity correction was used for each of the seven questions in relation to answering the question correctly and in attempting to answer the question. The independent variables were supplementary training group and class offering. Once problem independence for class was shown, the measures of correctness and attempts for problems favoring one group were tested by analysis of variance. Problem 1 furnished data for hypothesis 5 (dv 2), problem 2 for hypothesis 6 (dv 3), problems 3 and 4 for hypothesis 7 (dv 4), problem 5 for hypothesis 8 (dv 5), problem 6 for hypothesis 9 (dv 6), and problem 7 for hypothesis 10 (dv 7).
CHAPTER IV RESULTS

The purpose of this study was to determine if the difficulties students confront in indirect addressing can be attributable to a weakness in understanding algorithms or an inadequacy in handling indirect array notation. Specifically, the influence of two supplementary units of instruction on students' performance on supplying subscripts for indirect addressing and on procedural reasoning was examined.

The study was conducted using a posttest-only design. Responses to a questionnaire were used to provide a stratified random assignment of students to the supplementary instruction groups, NOTATION and ALGORITHM. The strata were previous computing experience, grade point average, and mathematics background. The students completed their computer-based supplementary instruction before they were presented lectures on the use of indirection with array accessing. After a weekend had passed from the time the lectures were presented, the students were administered two examinations. The first examination was a computer-based evaluation covering notational comprehension. The second examination was a paper-and-pencil evaluation assessing procedural reasoning.

A summary of the findings from the data is presented in this chapter. The chapter is divided into three major sections. In the first two sections, the findings from the
computer-based evaluation and the paper and pencil test are described. The final section contains a summary of the disposition of the hypotheses.

Summary of Findings from the Computer-based Evaluation

The computer-based evaluation contained problems in which the students from the Fall 1991 class were asked to supply numeric or symbolic subscripts involving zero, one or two levels (two versions, leftmost column, and any of the remaining columns) of indirect array notation. The students received four problems for each of the four levels, a total of 16 problems for both numeric subscripts and symbolic subscripts for a grand total of 32 problems. The evaluation was administered outside of the scheduled class time. The students were told their responses and response times would be recorded. A response in the allowed range of subscripts, either digits or designated symbols, was required by the program before the students could move forward to the next problem. The problems are listed in Appendix H. Students in the Fall 1990 class completed a similar exercise but the problems were randomly generated for each student so the scores were not comparable.

The descriptive statistics for the NOTATION and ALGORITHM groups are found in Table 1. The table is in two sections,
problems correct and problem response time, with two
subsections, digit and symbol subscripts. Each subsection is
split into two columns by supplementary training group,
NOTATION and ALGORITHM. Each subsection has four rows for
each level of indirection; none, one, two using left column
(Two left), and two using the remaining columns (Two rest).

Table 1  Computer-based Evaluation Descriptive Statistics for
Problems Correct and Problems Response Time

<table>
<thead>
<tr>
<th>Indirection level</th>
<th>Experimental groups</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Notation</td>
<td>Algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>SE</td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td>Numeric notation correct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>4.00</td>
<td>0.00</td>
<td>0.00</td>
<td>4.00</td>
<td>0.00</td>
</tr>
<tr>
<td>One</td>
<td>3.75</td>
<td>0.46</td>
<td>0.16</td>
<td>2.71</td>
<td>1.89</td>
</tr>
<tr>
<td>Two left</td>
<td>3.63</td>
<td>0.74</td>
<td>0.26</td>
<td>1.86</td>
<td>1.77</td>
</tr>
<tr>
<td>Two rest</td>
<td>3.50</td>
<td>0.53</td>
<td>0.19</td>
<td>2.14</td>
<td>1.68</td>
</tr>
<tr>
<td>Symbolic notation correct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>3.75</td>
<td>0.46</td>
<td>0.16</td>
<td>4.00</td>
<td>0.00</td>
</tr>
<tr>
<td>One</td>
<td>3.63</td>
<td>0.74</td>
<td>0.26</td>
<td>3.14</td>
<td>1.46</td>
</tr>
<tr>
<td>Two left</td>
<td>3.50</td>
<td>0.76</td>
<td>0.27</td>
<td>3.14</td>
<td>1.46</td>
</tr>
<tr>
<td>Two rest</td>
<td>3.75</td>
<td>0.46</td>
<td>0.16</td>
<td>3.43</td>
<td>0.54</td>
</tr>
<tr>
<td>Numeric response time (seconds)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>2.86</td>
<td>1.73</td>
<td>0.61</td>
<td>4.14</td>
<td>3.19</td>
</tr>
<tr>
<td>One</td>
<td>6.38</td>
<td>2.72</td>
<td>0.96</td>
<td>5.86</td>
<td>2.55</td>
</tr>
<tr>
<td>Two left</td>
<td>13.00</td>
<td>4.72</td>
<td>1.67</td>
<td>14.43</td>
<td>10.11</td>
</tr>
<tr>
<td>Two rest</td>
<td>10.13</td>
<td>2.30</td>
<td>0.81</td>
<td>13.29</td>
<td>7.20</td>
</tr>
<tr>
<td>Symbolic response time (seconds)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>5.00</td>
<td>2.20</td>
<td>0.78</td>
<td>3.57</td>
<td>1.51</td>
</tr>
<tr>
<td>One</td>
<td>9.00</td>
<td>2.83</td>
<td>1.00</td>
<td>11.29</td>
<td>6.92</td>
</tr>
<tr>
<td>Two left</td>
<td>11.63</td>
<td>3.77</td>
<td>1.34</td>
<td>16.86</td>
<td>6.52</td>
</tr>
<tr>
<td>Two rest</td>
<td>11.38</td>
<td>4.63</td>
<td>1.64</td>
<td>13.71</td>
<td>4.23</td>
</tr>
</tbody>
</table>

Note. Number of cases for Notation is 8 and for Algorithm 7.
It should be noted the NOTATION group had higher mean scores on the numeric notation exercises which were presented first. However, the means were nearly identical on the subsequent symbolic exercises, but the standard deviations were greater for the ALGORITHM group. By the time the students finished the symbol subscript portion of the evaluation, they were averaging at least three out of four problems correct with a mean response time of under 20 seconds. These results show the problems did require some thought but were not overwhelmingly difficult for either group.

Using an analysis of variance test to assess differences between instructional groups on the number of problems correct in using subscript notation, significant differences were found between group performances for numeric subscripts at two levels of indirection, both with the leftmost column and with the remaining columns, $F(1,14)=6.67, p < .05$, and $F(1,14)=4.74, p < .05$. These results are found in Table 2. The table is divided by the subscript type, numeric and symbolic. The groups were compared for each level of indirect array notation which appears as rows in the table.

Using the analysis of variance of the response times in using subscript notation for the instructional groups, no significant differences between the groups were found. The results of the analysis of variance tests appear in Table 3. This table is also divided by the subscript type, numeric and
symbolic, and the groups were compared for each level of indirect array notation. A reason for no significant

Table 2 ANOVA Test for Group Differences on Computer-based Evaluation for Problems Correct

<table>
<thead>
<tr>
<th>Indirection level</th>
<th>ANOVA statistics</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Source</td>
<td>df</td>
<td>MS</td>
</tr>
<tr>
<td>Numeric Subscripts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>Main effects</td>
<td>1</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>13</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>14</td>
<td>0.000</td>
</tr>
<tr>
<td>One</td>
<td>Main effects</td>
<td>1</td>
<td>4.005</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>13</td>
<td>1.764</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>14</td>
<td>1.924</td>
</tr>
<tr>
<td>Two left</td>
<td>Main effects</td>
<td>1</td>
<td>11.668</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>13</td>
<td>1.749</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>14</td>
<td>2.475</td>
</tr>
<tr>
<td>Two rest</td>
<td>Main effects</td>
<td>1</td>
<td>6.876</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>13</td>
<td>1.451</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>14</td>
<td>1.838</td>
</tr>
<tr>
<td>Symbolic Subscripts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>Main effects</td>
<td>1</td>
<td>0.233</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>13</td>
<td>0.115</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>13</td>
<td>0.124</td>
</tr>
<tr>
<td>One</td>
<td>Main effects</td>
<td>1</td>
<td>0.868</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>13</td>
<td>1.287</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>14</td>
<td>1.257</td>
</tr>
<tr>
<td>Two left</td>
<td>Main effects</td>
<td>1</td>
<td>0.476</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>13</td>
<td>1.297</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>14</td>
<td>1.238</td>
</tr>
<tr>
<td>Two rest</td>
<td>Main effects</td>
<td>1</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>13</td>
<td>0.247</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>14</td>
<td>0.257</td>
</tr>
</tbody>
</table>

* p < .05.
difference between groups is the small number of cases. However, since the differences between the means were

Table 3 Computer-based Evaluation ANOVA for Response Time

<table>
<thead>
<tr>
<th>Indirection level Source</th>
<th>ANOVA statistics</th>
<th>df</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric Subscripts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main effects</td>
<td>1</td>
<td>6.001</td>
<td>0.955</td>
<td></td>
</tr>
<tr>
<td>Residual</td>
<td>13</td>
<td>6.287</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>14</td>
<td>6.267</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main effects</td>
<td>1</td>
<td>1.001</td>
<td>0.143</td>
<td></td>
</tr>
<tr>
<td>Residual</td>
<td>13</td>
<td>6.979</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>14</td>
<td>6.552</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two left</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main effects</td>
<td>1</td>
<td>7.619</td>
<td>0.129</td>
<td></td>
</tr>
<tr>
<td>Residual</td>
<td>13</td>
<td>59.209</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>14</td>
<td>55.524</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two rest</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main effects</td>
<td>1</td>
<td>37.296</td>
<td>1.392</td>
<td></td>
</tr>
<tr>
<td>Residual</td>
<td>13</td>
<td>26.793</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>14</td>
<td>27.543</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symbolic Subscripts

| None                    |                  |    |      |      |
| Main effects            | 1                | 7.619 | 2.076 |
| Residual                | 13               | 3.670 |      |
| Total                   | 13               | 3.952 |      |
| One                     |                  |    |      |      |
| Main effects            | 1                | 19.505 | 0.738 |
| Residual                | 13               | 26.418 |      |
| Total                   | 14               | 25.924 |      |
| Two left                |                  |    |      |      |
| Main effects            | 1                | 102.201 | 3.745 |
| Residual                | 13               | 27.287 |      |
| Total                   | 14               | 32.638 |      |
| Two rest                |                  |    |      |      |
| Main effects            | 1                | 20.430 | 1.032 |
| Residual                | 13               | 19.793 |      |
| Total                   | 14               | 19.838 |      |
small for the numeric and symbolic notation, a conjecture can be made that choice of subscript notation is not a problem of consequence.

Summary of the Findings from the Paper-and-pencil Evaluation

The seven problem paper and pencil test (Appendix E) was administered immediately following the computer-based evaluation. It was completed by both the 1991 and 1992 classes and was scored for each item on whether the item was attempted and whether it was correct (as seen in Table 4). Since these classes were relatively small it was desirable to combine the data for purposes of analysis. Prior to doing so, however, a chi-square analysis was performed to determine if there were significant class differences on any of the seven problems. One significant difference was found in class performance on problem two. However, the performance between supplementary training groups on that problem did not differ significantly within each class so the data were combined. In other words, the 1991 class performed consistently with but at a significantly lower level than the 1990 class on problem two. The results of this analysis by class can be found in the upper section of Table 5.

The descriptive statistics for the problems on the paper and pencil test are shown in Table 4. This table is divided
vertically into two parts, problems answered correctly and problems attempted. The parts are split horizontally into two major columns for the supplemental training groups, NOTATION and ALGORITHM. Each part of the table has row entries for each of the seven problems. For every problem, a score of zero was assigned for an incorrect answer and a score of one for a correct answer.

Table 4 Paper and Pencil Evaluation Descriptive Statistics

<table>
<thead>
<tr>
<th>Problem</th>
<th>Mean</th>
<th>SD</th>
<th>SE</th>
<th>Mean</th>
<th>SD</th>
<th>SE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Notation</td>
<td></td>
<td></td>
<td>Algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>SE</td>
<td>Mean</td>
<td>SD</td>
<td>SE</td>
</tr>
<tr>
<td></td>
<td>Problems correct</td>
<td></td>
<td></td>
<td>Problems attempted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.68</td>
<td>0.47</td>
<td>0.10</td>
<td>0.96</td>
<td>0.21</td>
<td>0.04</td>
</tr>
<tr>
<td>2</td>
<td>0.41</td>
<td>0.50</td>
<td>0.11</td>
<td>0.52</td>
<td>0.51</td>
<td>0.11</td>
</tr>
<tr>
<td>3</td>
<td>0.68</td>
<td>0.47</td>
<td>0.10</td>
<td>0.78</td>
<td>0.42</td>
<td>0.09</td>
</tr>
<tr>
<td>4</td>
<td>0.14</td>
<td>0.35</td>
<td>0.08</td>
<td>0.13</td>
<td>0.34</td>
<td>0.07</td>
</tr>
<tr>
<td>5</td>
<td>0.18</td>
<td>0.40</td>
<td>0.08</td>
<td>0.13</td>
<td>0.34</td>
<td>0.07</td>
</tr>
<tr>
<td>6</td>
<td>0.82</td>
<td>0.40</td>
<td>0.08</td>
<td>0.65</td>
<td>0.49</td>
<td>0.10</td>
</tr>
<tr>
<td>7</td>
<td>0.27</td>
<td>0.46</td>
<td>0.10</td>
<td>0.22</td>
<td>0.42</td>
<td>0.08</td>
</tr>
<tr>
<td>1</td>
<td>0.77</td>
<td>0.43</td>
<td>0.09</td>
<td>0.96</td>
<td>0.21</td>
<td>0.04</td>
</tr>
<tr>
<td>2</td>
<td>0.41</td>
<td>0.50</td>
<td>0.09</td>
<td>0.70</td>
<td>0.47</td>
<td>0.10</td>
</tr>
<tr>
<td>3</td>
<td>0.68</td>
<td>0.48</td>
<td>0.10</td>
<td>0.83</td>
<td>0.39</td>
<td>0.08</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>0.51</td>
<td>0.11</td>
<td>0.52</td>
<td>0.51</td>
<td>0.11</td>
</tr>
<tr>
<td>5</td>
<td>0.73</td>
<td>0.46</td>
<td>0.10</td>
<td>0.91</td>
<td>0.29</td>
<td>0.06</td>
</tr>
<tr>
<td>6</td>
<td>0.91</td>
<td>0.29</td>
<td>0.06</td>
<td>0.87</td>
<td>0.34</td>
<td>0.07</td>
</tr>
<tr>
<td>7</td>
<td>0.32</td>
<td>0.48</td>
<td>0.10</td>
<td>0.70</td>
<td>0.47</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Notes. Number of cases for Notation is 22 and for Algorithm is 23. Problem 1, 2, 3, 4, and 7 were hypothesized to favor the Algorithm group.
From this table it can be seen that 70 percent or more of the students in the ALGORITHM group attempted all problems except problem four which was attempted by only 52 percent of the students. In contrast, in the NOTATION group, only 41 percent of the students attempted problem two, 50 percent problem four and 32 percent problem seven. It should also be noted that problems one, three, and six were answered correctly by most of the students in both groups and problem two was answered correctly by about half of the students; 41 percent in the NOTATION group and 52 percent in the ALGORITHM group. Problem four, five and seven were much more difficult. They were answered correctly by only 14, 18 and 27 percent of the NOTATION group and 13, 13 and 22 percent of the ALGORITHM group respectively.

To determine if there were significant differences on any of the seven problems for a correct answer or an attempted answer solely due to group, a chi-square test of independence with Yates continuity correction for (0,1) values was used and appears in the lower section of Table 5. Two significant differences were found. On problem one, in which students were required to change bubble sort code which initially moved the smallest element to the beginning of the array to code which initially moved the largest number to the end of the array, the ALGORITHM group demonstrated superior performance, \( \chi^2 = 4.04, p < .05 \). The mean for the Algorithm group was .96
and for the NOTATION group .68. On problem seven, in which students were required to explain how different results could be obtained from bubble and selection sorts, there was a difference in the number of solutions attempted, $\chi^2 = 4.99$, $p < .05$. Seventy percent of the ALGORITHM students attempted this problem whereas it was attempted by only 32 percent of the NOTATION students.

The correct problem scores of problems one, two, three,

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Correct $\chi^2$ value</th>
<th>Correct $p$</th>
<th>Attempted $\chi^2$ value</th>
<th>Attempted $p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>1.00</td>
<td>0.21</td>
<td>0.64</td>
</tr>
<tr>
<td>2</td>
<td>12.15</td>
<td>0.00&quot;</td>
<td>5.95</td>
<td>0.01&quot;</td>
</tr>
<tr>
<td>3</td>
<td>0.72</td>
<td>0.72</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>0.22</td>
<td>0.64</td>
<td>0.01</td>
<td>0.91</td>
</tr>
<tr>
<td>5</td>
<td>0.53</td>
<td>0.47</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>0.13</td>
<td>0.72</td>
<td>0.70</td>
<td>0.40</td>
</tr>
<tr>
<td>7</td>
<td>0.02</td>
<td>0.90</td>
<td>0.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

For independence of group:

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Correct $\chi^2$ value</th>
<th>Correct $p$</th>
<th>Attempted $\chi^2$ value</th>
<th>Attempted $p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.08</td>
<td>0.04&quot;</td>
<td>1.89</td>
<td>0.17</td>
</tr>
<tr>
<td>2</td>
<td>0.21</td>
<td>0.65</td>
<td>2.67</td>
<td>0.10</td>
</tr>
<tr>
<td>3</td>
<td>0.18</td>
<td>0.67</td>
<td>0.61</td>
<td>0.43</td>
</tr>
<tr>
<td>4</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
<td>0.95</td>
<td>1.54</td>
<td>0.22</td>
</tr>
<tr>
<td>6</td>
<td>0.85</td>
<td>0.35</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>0.01</td>
<td>0.93</td>
<td>4.99</td>
<td>0.03&quot;</td>
</tr>
</tbody>
</table>

Note. Degree of freedom is 1.

' $p < .05$.
" $p < .01$. 

Table 5 $\chi^2$ Test with Yates Continuity Correction on Paper and Pencil Evaluation.
four, and seven were summed. Also the attempted problem scores of the same problems were summed. These problems were hypothesized to favor the ALGORITHM group. An analysis of variance by group was performed on these sums which appears in Table 6. Likewise, scores from problems five and six, which favored NOTATION, were summed. Another pair of analysis of variance tests by group were performed. The only significant difference between groups was detected on problems attempted which were hypothesized to favor ALGORITHM, $F(1,44)=4.62$.

<table>
<thead>
<tr>
<th>Problem Group Source</th>
<th>ANOVA statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>df</td>
<td>MS</td>
</tr>
<tr>
<td>Problems Correct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main effects</td>
<td>1</td>
<td>2.049</td>
</tr>
<tr>
<td>Residual</td>
<td>43</td>
<td>1.878</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>1.882</td>
</tr>
<tr>
<td>Notation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main effects</td>
<td>1</td>
<td>0.531</td>
</tr>
<tr>
<td>Residual</td>
<td>43</td>
<td>0.324</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>0.328</td>
</tr>
<tr>
<td>Problems attempted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main effects</td>
<td>1</td>
<td>11.558</td>
</tr>
<tr>
<td>Residual</td>
<td>43</td>
<td>2.503</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>2.709</td>
</tr>
<tr>
<td>Notation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main effects</td>
<td>1</td>
<td>0.240</td>
</tr>
<tr>
<td>Residual</td>
<td>43</td>
<td>0.349</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>0.346</td>
</tr>
</tbody>
</table>

* $p < .05$. 

Table 6 Paper and Pencil Evaluation ANOVA of Groups on Problems Favoring One Group
p < .05. The ALGORITHM group with a mean of 0.96, attempted more problems than the NOTATION group, with a mean of 0.77.

Testing the Hypotheses

The major goal of this study was to determine which treatment, if any, removed student difficulties with indirect addressing. In view of this goal and based upon the findings previously presented in this chapter, the rejection or lack thereof of the ten hypotheses stated in Chapter 1 will now be considered.

Hypothesis 1. The NOTATION group will perform significantly better than the ALGORITHM group in correctly supplying a numeric subscript (dv 1a).

This hypothesis was tested using an analysis of variance test to determine if there were significant differences between the scores of the NOTATION and ALGORITHM groups on supplying correct numeric subscripts for each of the four levels of the computer-based evaluation. Because this activity was very similar to the training received by the students in the NOTATION group it was hypothesized that they would achieve higher mean scores. Although the means for the NOTATION group were higher, there were no significant
differences between the groups on the first two levels. On the last two levels in which groups manipulated two levels of indirect array notation, accessing the leftmost column and accessing any other column, the NOTATION group performed significantly better than the ALGORITHM group. With these results, directional Hypothesis 1 was accepted.

Hypothesis 2. The NOTATION group will perform significantly better than the ALGORITHM group in the time required to supply numeric subscripts (dv la).

The response times for supplying a subscript were recorded for each student on each of the numeric problems on the computer-based evaluation. These times were analyzed using the analysis of variance for each of the four levels to determine if there were differences between the experimental groups. Since no significant differences were found on any level the null hypotheses were not rejected. There was insufficient evidence to accept directional Hypothesis 2.

Hypothesis 3. The NOTATION group will perform significantly better than the ALGORITHM group in correctly supplying a character subscript (dv lb).
The scores for the number of correct responses on the four levels of the symbolic portion of the computer-based evaluation were analyzed using an analysis of variance to test this hypothesis. The means for the NOTATION group were only slightly higher on these measures, producing no significant differences on any of the tests. Thus the null hypotheses were not rejected and directional Hypothesis 3 was not accepted.

Hypothesis 4. The NOTATION group will perform significantly better than the ALGORITHM group in the time required to supply character subscripts (dv lb).

As with the numeric portion of the computer-based evaluation response times were recorded for the symbolic exercises. The analysis of variance tests on this data for each of the four levels of subscripting revealed no significant differences. Since the null hypotheses were not rejected, there was insufficient evidence to accept directional Hypothesis 4.

The data for disposing of Hypotheses 5 through 10 were obtained from the seven problems of the paper and pencil test. Problems one, two, five, six, and seven supplied data for Hypotheses 5, 6, 8, 9 and 10 respectively. Problems three and
four supplied data for Hypothesis 7. For each problem a chi-square test was performed on the number of correct answers and number of problems attempted. For these tests the data from the Fall 1990 and Fall 1991 classes were combined.

Hypothesis 5. The ALGORITHM group will perform significantly better than the NOTATION group in correctly reversing the comparison direction of a bubble sort (dv 2).

Problem one asked the students to rewrite the code for a bubble sort to reverse the order in which the array was filled. The original code filled the array from top to bottom and the correctly revised code filled the array from bottom to top. Students who understood the algorithm being used could solve this problem without changing subscripts or even considering problems of indirection. As was predicted, the chi-square test for correct solutions revealed a significant difference between the test groups favoring the ALGORITHM group. Although more students in the ALGORITHM group attempted this problem no significant differences were found on that variable. On the basis of this analysis, a null hypothesis was rejected and directional Hypothesis 5 was accepted.
Hypothesis 6. The ALGORITHM group will perform significantly better than the NOTATION group in correctly adding an efficient terminator for the bubble sort (dv 3).

Problem two required the code for the bubble sort in problem one to be modified to halt the sort once the data was in order. As in the case of problem one, the solution of this problem was primarily dependent on understanding of the sorting algorithm and independent of indirection. However, unlike problem one, no significances were found between the groups either for number correct of number attempted. Directional Hypothesis 6 was not accepted.

Hypothesis 7. The ALGORITHM group will perform significantly better than the NOTATION group in correctly reversing the replacement location of a selection sort with and without indirection (dv 4).

The disposition of Hypothesis 7 was based on data from problems three and four. Problem three was similar to problem one except the code supplied was for a selection algorithm. The students were to reverse the order in which the array was filled. Tests of the null hypotheses revealed no significant differences for either the number correct or problems
attempted. Problem four required the students to remove the indirection from the selection code and to sort the array directly. This involved an understanding of the selection algorithm as well as the relationship between the array of addresses and the array of data. Students in both groups performed very poorly on this problem. The chi-square tests revealed no significant differences between the groups. On the basis of this analysis there was insufficient evidence to accept directional Hypothesis 7.

Hypothesis 8. The NOTATION group will perform significantly better than the ALGORITHM group in correctly identifying the distance between the first and last elements in an array given only the ordered indirect array (dv 5).

Problem 5 pictured a sorted indirect array of values and its array of data with the values missing. The students were to determine the number of elements between the largest and smallest elements in the array of data. Answering this question correctly required an understanding of the relationship between the indirect array and the data array. Few students in either group correctly answered this problem. Tests of the null hypothesis revealed no significant
differences. Thus the directional Hypothesis 3 was not accepted.

Hypothesis 9. The NOTATION group will perform significantly better than the ALGORITHM group in correctly supplying the inverse of a given indirect array (dv 6).

The data for testing Hypothesis 9 was obtained from problem 6. The problem presented two indirect arrays with the elements shown only for the first. The students were to supply elements for the second array so that the indirect array names could be entered in either order to access the same element of the data array. This problem required an understanding of indirect notation and use of analytical skill in solving a logic problem. Knowledge of sorting algorithms does not appear to be useful in solving this problem. The mean for the NOTATION group was higher than the mean for the ALGORITHM group on the number of correct solutions but no significant differences were found between the groups for either number correct or number attempted. Directional Hypothesis 9 was not accepted.
Hypothesis 10. The ALGORITHM group will perform significantly better than the NOTATION group in correctly identifying why the indirect arrays from bubble and selection sort are different (dv 7).

Problem 7 of the paper and pencil test required an explanation for different indirect arrays resulting from a bubble and selection sort of the same data. This problem required a strong understanding of both sorting algorithms and probably careful analysis of several examples. It was quite difficult with only about one fourth of the students answering it correctly. Tests of the null hypotheses for number correct revealed no significant differences; however, differences were found between the groups on the number of responses. More than twice as many students in the ALGORITHM group attempted this problem. In light of this difference, directional Hypothesis 10 was accepted.
CHAPTER V DISCUSSION

There were two fundamental objectives of this study. The first was to better understand the difficulties students have in learning indirect addressing. The second was to determine more effective ways to teach the concept of indirection to students.

The study was conducted on beginning programming students by randomly dividing the students between two supplementary training groups. One supplementary training unit was designed to be a computer-based tutorial on sort algorithm development. The other supplementary training unit was intended to be a computer-based simulation program using indirect array notation. A posttest only experimental design was used.

The study attempted to measure student comprehension differences on two levels. The first level involved application of indirect array notation. The second level assessed success in modifying sort routines, which incorporated indirection, and understanding of indirect array manipulations.

Because of the explorative nature of this study and the small sample size, the findings must be viewed as tentative. However, it would appear indirect array notation by itself is not an important source of student problems. In fact, there was little evidence to suggest either supplementary training
unit made a sizeable difference in the student’s ability to deal with indirection within the context of programs. There was an indication extra study of algorithms encouraged students to attempt to solve more problems, but this finding may be a result of the experimental conditions and may not be generalizable. These findings and other aspects of the study are discussed in the next section.

Discussion

This section is subdivided into discussions of student performance on the computer-based evaluation and the paper and pencil evaluation. Following these discussions is a brief analysis of the NOTATION unit.

Computer-based Evaluation

Since the computer-based evaluation was a modification of part of the NOTATION lesson, differences in the performance between the supplementary training groups were anticipated. The NOTATION group had practiced similar exercises in addition to performing tasks which were designed to strengthen their understanding of the use of indirection. Because of this experience and the perceived difficulty of using indirect
notation, it was anticipated the NOTATION group would achieve significantly higher scores.

On the initial numeric subscript problem, which required no indirection, all students achieved a perfect score indicating a basic understanding of using subscripted variables. On the next problem sets which required indirection, the performance of the NOTATION group was better than the ALGORITHM group in number of problems answered correctly and response time. In fact, there were significant differences in correctly answering problems involving two levels of indirect array notation. What was not anticipated, however, was the relative ease with which the ALGORITHM students mastered this exercise. It only required the ALGORITHM students one cycle of 16 problems to achieve nearly the same average level of competency as the NOTATION group. The performance of the two groups was not significantly different on the symbolic notation problems. Yet, the standard deviation for the ALGORITHM group was higher, possibly indicating that lower ability or less experienced students profited from the NOTATION activities. This conjecture would require a much greater sample of low ability students to substantiate.

In failing to find lasting differences between the groups on this measure, it would appear that using indirect notation was not as difficult a concept as originally thought.
However, it is possible that using indirect notation and having a deep understanding of the concept are not synonymous or the evaluation may not have tapped the extent of the effects of the NOTATION experience. It is also possible the students were of sufficiently high ability that the NOTATION experience did not force restructuring of their schema even though the schema may have been inadequate. Resolution of these questions is beyond the scope of this study.

**Paper and Pencil Evaluation**

Prior to analyzing the data from this instrument, a decision had to be made concerning the combining of the data from the two classes. With the small number of participants it was strongly desired to treat the data as a single sample. Prior to doing so, an analysis comparing the classes on each of the seven problems was performed. The only occurrence of significant difference in class performance was found on problem two. This problem asked the students to modify the bubble sort algorithm so that the algorithm would stop sorting once the data were in order. Only one student from the 1991 class correctly answered the problem whereas approximately half of the students in each group from the 1990 class produced correct answers. This difference may reflect a slight difference in content covered in the two courses.
Three kinds of questions were contained in the paper and pencil evaluation. The first four questions were based on sorting algorithms and required making changes to existing code. The next two problems involved indirect addressing independent of code or algorithms. The final problem required an insightful analysis of both selection- and bubble sort routines. On the whole, the evaluation was time consuming and challenging.

The organization of this evaluation and its length may have been to the disadvantage of the NOTATION group. The first four problems could be solved most efficiently by completely ignoring the indirect notation and concentrating on the algorithms. In fact, focusing on the notation would have been a major distraction in reaching the assigned goal. Beyond the immediate disadvantage, if the students in the NOTATION group worked through the indirection in the code presented, they would have used time and energy which would have detracted from their success on later problems.

Even the placement of the paper and pencil evaluation after the computer-based evaluation could have been important. Since the ALGORITHM group was so successful on the final problems of the computer-based evaluation, they effectively received both supplementary training units. Thus the study may not have compared ALGORITHM versus NOTATION but ALGORITHM and NOTATION versus NOTATION only. When consideration is
given to these factors, the superior performance of the ALGORITHM group is not surprising. In fact, it is somewhat surprising this group did not achieve greater success on problems four, five and seven. Those problems merit a closer inspection.

The fourth problem was to write a selection sort routine which did not involve indirection. This routine should have been familiar to all students because it was the first sorting procedure to which they had been introduced in the regular class lectures. Yet, this problem revealed the worst performance by both groups in providing correct answers. Only half of the NOTATION students and a little more than half of the ALGORITHM students attempted the problem. As stated in the evaluation, the problem could be solved by removing the indirection from two sections of the code. Upon examination of the incorrect responses, 22 percent of the ALGORITHM group had one of the two sections correct whereas 75 percent of the NOTATION group had one of the two sections correct. While this difference was not statistically significant, the lower performance by the ALGORITHM group may indicate they had been ignoring indirection in solving the first three problems.

Problem five may provide another indicator indirection was being ignored by the students. The objective of this problem was to determine the number of data elements physically located between the logical first and last elements
of the data array. The answer the students gave was correct for the indirect array but not correct for the data array. It is somewhat surprising that so many of the NOTATION group failed to answer this problem correctly.

Problem seven required the students to explain why the indirect arrays from two sort routines were different. This problem required a deep understanding of the nature of the two routines so it was speculated the supplementary training units would not be sufficient to provide an answer to the problem. The descriptive statistics for the problem show while 70 percent of ALGORITHM students attempted the problem (significantly more than NOTATION students) only 22 percent got the problem correct, whereas, 32 percent of the NOTATION students attempted the problem but 27 percent got the problem correct. Thus, nearly every NOTATION student attempting the problem solved it. In this case, it might have been advantageous to conduct an exit interview with the students in the two groups to determine if their way of approaching the problem was different and if the strategy could be traced to some aspect of the training. This problem could lead to a more revealing evaluative instrument.

A final observation from this instrument is the overall effect of the ALGORITHM unit. The material covered in the unit was not different from the material covered in the handout or the lecture. With respect to content, it could
best be described as more of the same. There was, however, one aspect of the unit that may have made a difference. It actively demonstrated the execution of the sort algorithms under the control of the student. Watching the process execute may have been advantageous.

Experiencing Simulation

One of the motivating factors behind this research was to gain knowledge about student difficulties with indirection which would enable the creation of an experiencing simulation. The NOTATION unit was intended to provide much of the evidence. It was expected that student experience gained from this unit would produce performance differences between the groups and reveal deficiencies in understanding. By studying those differences, underlying causes of difficulties in using indirect ion should have been evident. Unfortunately, the NOTATION unit did not make the expected contribution. It did provide another instance of a direct treatment failing to contribute to the solution of a complex learning problem.

Thomas and Lewis (1990) described an experiencing simulation in computer science as "a computer program containing a graphic model of selected aspects of a computer system. The program enables the student to change the model from a given state, through several intermediate states, to a
specified goal state. The program accepts commands from the student, alters the state of the model, and when appropriate displays the new state." The model must include key aspects of the content to be learned as determined by the learner's needs and the commands must permit insightful manipulation of that content. The activities must be chosen to insure that the student interacts with the content in a challenging and meaningful way. Determining the content to include in the model, the commands to provide the learner, and the activities to assign requires a deep understanding of the learning difficulties inherent in the topic to be mastered.

The NOTATION unit contained a model of indirect notation and activities for the student to master in order to use this technique. The ease with which the ALGORITHM group appeared to acquire the specified skill and the lack of success of both groups in using the technique in context indicates the underlying causes of student difficulty were not removed. The weakness of the NOTATION unit may reside in its emphasis of direct application and its failure to contribute the development of procedural reasoning. An experiencing simulation would minimize the syntax and direct application of techniques and maximize exposure to connections between the problems to be solved and the processes for solving them. While this study does little to define the experiencing simulation it does support the need for one.
Based on this study, six recommendations are made concerning the improved acquisition of procedural reasoning.

First, rearranging the type of problems found in the paper and pencil evaluation should be done. A pattern of ignoring the indirection should not be promulgated in the evaluation problems. Possibly, problems containing indirect notation which involve algorithms such as merging or searching should be included.

Second, students should be required to provide answers to all problems. Without an answer to each problem the student’s thought processes cannot be adequately studied. One way of moving toward this goal would be to make the unit a required part of the course.

Third, performing a modified version of the same study on a larger sample of students in their second semester of programming might increase the revelations of the evaluations. The students would have more programming experience and might be more motivated to participate in the study. They would also have been exposed to more algorithms of a similar complexity to the sort routines used in the study.

Fourth, further investigations using more problems of the nature of problem seven on the paper-and-pencil evaluation would be useful. Using indirection as a compass to detect
deeper understanding in programming concepts might help uncover misconceptions about those programming concepts. Exit interviews could also reveal additional information about the student’s thinking process.

Fifth, the differences of the standard deviations exhibited by students in the computer-based evaluation needs closer examination. By removing the high ability students from the sample, the effects of the supplementary training units might be more greatly seen.

Sixth, the student difficulty which appeared on paper-and-pencil problem four of removing the indirect array notation bears more scrutiny. Because the students were trained on this algorithm, the cause of their difficulty in applying their knowledge when the problem was camouflaged could be important. Factors such as this often lead to components of experiencing simulations.

Conclusion

The work of Sheil and Mayer has begun the reexamination of the methods used to teach programming. Instead of using a natural language or mathematics model which focuses on programming on a micro scale, syntax and programming plans, their work has brought attention to programming on a macro scale, procedural reasoning. The work of Thomas and others on
the use of experiencing simulation to overcome the traditional
hierarchical teaching pattern of programming has offered an
opportunity to study and develop procedural reasoning skills
in ways not before possible because the computer is the active
medium.

These studies focused on the importance of what the
student cognitively brings to a learning activity. By better
understanding the difficulties faced by students learning
programming concepts, learning activities can be better
constructed to anticipate those difficulties and possibly
remove them.

The need for student proficiency with indirection is
vital in the process of the student becoming an expert
programmer or computer scientist. The aim of this study was
to explore the nature of student difficulties with one aspect of
indirection, that of indirect array notation. With the
qualifications of small sample size and the explorative nature
of the study, the major finding of this study was student
comprehension of indirect array notation by itself was not an
importance source of student difficulties. It is hoped with
further study of the concept of indirection, the root cause(s)
of problems will be found and thereby change the character of
programming instruction from an ad hoc approach to one which
is cognitively sound.
REFERENCES


ACKNOWLEDGEMENTS

I wish to thank Dr. Rex Thomas who has an abiding interest in the discovery of how students think about a problem. His work presented me with a unique opportunity to combine my interests in computer science and teaching.

I also want to acknowledge my friends and colleagues at Northwest Missouri State University and Mount Mercy College who gave me encouragement to preserve as I taught fulltime. I am especially grateful that my colleagues at Central College who allowed me to use their students as subjects of my study.

Last and most important, I wish to thank my wife, Renae, who has seen me through all the highs and lows of this process. Without her presence and prodding, this event would not the same meaning as it does for me now.
CONSENT FORM

You have agreed to evaluate a computer-based lesson that deals with indirect addressing using arrays. The lesson deals with an important part of this course. The lesson itself has not been formally tested. Thus, I am asking you to help me decide the best form and use of this lesson.

You will be asked to work through the lesson as a homework assignment. For this assignment, the class will be divided into two groups with each group working through a different version of the lesson. To judge the effectiveness of each version you will be asked to solve some problems using what you have learned from the lesson and from the class lectures which follow the lesson. The lesson will take approximately 1/2 hour and the problems will take about one hour. You will also be asked to express your opinion on the usefulness of various aspects of the lesson and to make any suggestions for improvement.

Your participation in the evaluation of the lesson and the data collected will not affect your grade in this course. Also, any reports of this study will not include identifying information which could be traced to a specific individual.

The purpose of this study is to help me improve my ways of presenting computing concepts. Since you are a volunteer, you may decline from doing the lesson and evaluation without penalty. Your participation adds to the insight I can gain. If you have any questions about either the lesson or evaluation please feel free to ask.

I ask for your signature below to indicate your willingness to allow the data collected during this activity to be used in evaluating and reporting the effectiveness of this lesson.

Thank you for your cooperation.

( Signature of the Participant )
BACKGROUND QUESTIONNAIRE

Student No.

1. Age ____

2. Gender M F

3. GPA ____

4. Why are you taking this course?
   ____ Required for my major/minor
   ____ Other - Please specify

5. Have you had any previous computer programming courses? If so, what were they and when did you have them?

6. Please mark those math courses that you have had or are currently taking.
   ____ Calculus II
   ____ Calculus I
   ____ Discrete Math
   ____ Pre-calc/Trig/Math Analysis
   ____ Algebra II
   ____ Geometry
APPENDIX B

COMPUTER-BASED MATERIALS
LESSON IndirectA
SET FKEY,TERMINATE
SET KEYPAD,APPLICATION
CCOLOR ALL
FCOLOR GREEN,1
;LOG FULL
DEFINE Pos[5]:STRING $$ Used in units Databox, 8-11
DEFINE Name[5]:STRING $$ Ibid
DEFINE Age[5]:STRING $$ Ibid
DEFINE W[5]:STRING $$ Used in InitialGrid, 11-17
DEFINE X[5]:INTEGER $$ Ibid
DEFINE Y[5]:INTEGER $$ Ibid
DEFINE Z[5]:INTEGER $$ Ibid
DEFINE DELAY:REAL $$ Delay between pauses
DEFINE DONE:BOOLEAN $$ Used as loop control in 11-17
DEFINE TALLY[4,2]:INTEGER $$ Used in units InitialGrid, 11-17
DEFINE I:INTEGER
FOR I:=1,4
  ASSIGN TALLY[I,1]:=0
  ASSIGN TALLY[I,2]:=0
ENDFOR
DEFINE Line[8]:STRING $$ Used in units 2-5
ASSIGN Line[1] :="a Mary 42"
ASSIGN Line[2] :="b George 29"
ASSIGN Line[3] :="c Ilsa 37"
ASSIGN Line[4] :="d Tomas 63"
ASSIGN Line[5] :="e Tumara 14"
ASSIGN Line[6] :="f Akbar 50"
ASSIGN Line[7] :="g Elaine 19"
ASSIGN Line[8] :="11"
ASSIGN DELAY:=2.0
OPEN "Treat.dat",4,WRITE,SEQUENTIAL
DO One
CLOSE 4
DO Two
DO Three
DO Four
DO Five
DO Six
DO Seven
DO Eight
DO Nine
DO Ten
DO Eleven
DO Twelve
ASSIGN DELAY:=1.0
DO Thirteen
UNIT Return
AT 2255
WRITE press RETURN to continue
BOX 2261;2367
PAUSE
ERASE 2255;2383

UNIT EnterMessage
MODE INVERSE
AT 2205
WRITE Press the KEYPAD ENTER key to exit.
MODE NORMAL

UNIT Unhilite( I, Str )
DEFINE Str:STRING
DEFINE I:INTEGER
MODE REPLACE
MODE NORMAL
ERASE 0512+(I*100);0628+(I*100)
AT 0512+(I*100)
WRITE <<S,Str>>

UNIT Hilite( I, Str )
DEFINE Str:STRING
DEFINE I:INTEGER
MODE REPLACE
MODE INVERSE
ERASE 0512+(I*100);0628+(I*100)
AT 0512+(I*100)
WRITE <<S,Str>>
MODE NORMAL
UNIT One
SIZE 5
AT 0215
WRITE Welcome to
    Array
    Indirect
SIZE 1
DO Return
ERASE
AT 0515
WRITE Please enter your student number.
INPUT *
PUT 4,RESPONSE
ERASE
AT 0515
WRITE This lesson will introduce the concept of indirect
    access with the use of arrays.
PAUSE ELAPSED,DELAY
AT 1015
WRITE But first, let us review searching for items within
    a list.
DO Return
ERASE
UNIT Two
DEFINE I:INTEGER
DEFINE Notdone:BOOLEAN
AT 0210
WRITE Given a list of seven names and ages:
PAUSE ELAPSED,DELAY
AT 0410
WRITE Person Name Age
LINE 0510;0516
LINE 0518;0524
LINE 0526;0529
FOR I:=1,7
  . AT 0512+(1*100)
  . WRITE «S,LINE[ I ]»
ENDFOR
AT 1510
WRITE Using the up-arrow and down-arrow keys highlight the
person who is the THIRD YOUNGEST and then press the
return key.
DO Hilite( 8, Line[ 8 ] )
ASSIGN I:=8
ASSIGN Notdone:=TRUE
PROMPT ""
LOOP Notdone
  . ERASE 1800;1983
  . AT 1800
  . INPUT *
  . TEST response
  . VALUE "[UPA_KEY]" $$UP arrow key pressed
  . . DO Unhilite( I, Line[ I ] )
  . . IF (I-1)<1
  . . . ASSIGN I:=7
  . . ELSE
  . . . ASSIGN I:=I-1
  . . ENDIF
  . . DO Hilite( I, Line[ I ] )
  . . VALUE "[DNA_KEY]" $$DOWN arrow key pressed
  . . DO Unhilite( I, Line[ I ] )
  . . IF (I+1)>7
  . . . ASSIGN I:=1
  . . ELSE
  . . . ASSIGN I:=I+1
  . . ENDIF
  . . DO Hilite( I, Line[ I ] )
  . OTHER
  . . IF KEYPRESSED=13 $$RETURN pressed
  . . . DO Unhilite( I, Line[ I ] )
  . . . IF I=2
  . . . . AT 1810
  . . . . WRITE That is correct!
ELSE

AT 1810
WRITE That is not correct.
AT 2010
WRITE The correct answer is
DO Hilite( 2, Line[ 2 ] )
DO PAUSE ELAPSED,DELAY
DO Unhilite( 2, Line[ 2 ] )
ENDIF
ASSIGN Notdone:=FALSE
ELSE
AT 1810
WRITE Please press the up arrow, down arrow
or return key.
PAUSE ELAPSED,1.0
ENDIF
ENDTEST
ENDLOOP
PAUSE ELAPSED,DELAY
ERASE 1510;2183
AT 1510
WRITE Again using the arrow keys, highlight the person
whose name would appear SECOND if the names were
appearing in a dictionary.
DO Hilite( 8, Line[ 8 ] )
ASSIGN I:=8
ASSIGN Notdone:=TRUE
PROMPT ""
LOOP Notdone
ERASE 1800;1983
AT 1800
INPUT *
TEST response
VALUE "[UPA_KEY]" $$UP arrow key pressed
DO Unhilite( I, Line[ I ] )
IF (I-1)<1
ASSIGN I:=7
ELSE
ASSIGN I:=I-1
ENDIF
DO Hilite( I, Line[ I ] )
VALUE "[DNA_KEY]" $$DOWN arrow key pressed
DO Unhilite( I, Line[ I ] )
IF (I+1)>7
ASSIGN I:=1
ELSE
ASSIGN I:=I+1
ENDIF
DO Hilite( I, Line[ I ] )
. OTHER
. . IF KEYPRESSED=13 $RETURN pressed
. . . DO Unhilite( I, Line[ I ] )
. . . IF I=7
. . . . AT 1810
. . . . WRITE That is correct!
. . . . ELSE
. . . . . AT 1810
. . . . . WRITE That is not correct.
. . . . . AT 2010
. . . . . WRITE The correct answer is <<S,Line[ 7 ]>>
. . . . . DO Hilite( 7, Line[ 7 ] )
. . . . . PAUSE ELAPSED,DELAY
. . . . . DO Unhilite( 7, Line[ 7 ] )
. . . . ENDIF
. . . . ASSIGN Notdone:=FALSE
. . . ELSE
. . . . AT 1810
. . . . . WRITE Please press the up arrow, down arrow or return key.
. . . . . PAUSE ELAPSED,1.0
. . . . ENDIF
. ENDTEST
ENDLOOP
PAUSE ELAPSED,DELAY
UNIT QOne(I,Try,S)  $$ Used in units 3 and 4
DEFINE  I:INTEGER
DEFINE   Try:INTEGER
DEFINE   S:STRING
ASSIGN   QLENGTH:=1
QUERY    0543+(I*100)
RIGHTV   S
  .  MODE NORMAL
  .  AT  0550+(I*100)
  .  WRITE  This is correct!
  .  DO  Return
  .  ERASE  0550+(I*100);0666+(I*100)
  .  ASSIGN   Try:=0
WRONG
  .  IF Try=2
  .   .  MODE NORMAL
  .   .  AT  0550+(I*100)
  .   .  WRITE  The correct answer is <<S,S>>
  .   .  DO  Return
  .   .  ERASE  0550+(I*100);0673+(I*100)
  .   .  ASSIGN   Try:=0
  .   .  JUDGE   STOP
  .  ELSE
  .   .  MODE NORMAL
  .   .  AT  0550+(I*100)
  .   .  WRITE  Re-examine the data and try again
  .   .  PAUSE  ELAPSED,DELAY
  .   .  ERASE  0550+(I*100);0683+(I*100)
  .   .  MODE INVERSE
  .   .  ASSIGN   Try:=(Try+1)
  .  ENDIF
ENDQ
UNIT Three
DEFINE Try: INTEGER
ASSIGN Try := 0
DEFINE Young[7]: STRING
ASSIGN Young[1] := "e"
ASSIGN Young[2] := "g"
ASSIGN Young[3] := "b"
ASSIGN Young[4] := "c"
ASSIGN Young[5] := "a"
ASSIGN Young[6] := "f"
ASSIGN Young[7] := "d"
DEFINE I: INTEGER
ERASE
AT 0410
WRITE Person Name Age Youngest
LINE 0510; 0516
LINE 0518; 0524
LINE 0526; 0529
LINE 0540; 0548
LINE 1440; 1446
AT 1440
WRITE Oldest
FOR I := 1, 7
  AT 0512 + (I * 100)
  WRITE "<S,LINEx [{ I }>>
ENDFOR
AT 1510
WRITE Using the letter under the Person column, rank these people from youngest to oldest.
PROMPT ""
ASSIGN I := 1
LOOP I<8
  MODE INVERSE
  AT 0543 + (I * 100)
  WRITE
  DO QOne(I, Try, Young[I])
  ERASE 0543 + (I * 100); 0644 + (I * 100)
  AT 0543 + (I * 100)
  WRITE "<S, Young[I]>>
  ASSIGN I := (I + 1)
ENDLOOP
DO Return
UNIT Four
DEFINE Reverse[7]:STRING
ASSIGN QLENGTH:=1
ASSIGN Reverse[1]:="e"
ASSIGN Reverse[2]:="d"
ASSIGN Reverse[3]:="a"
ASSIGN Reverse[4]:="c"
ASSIGN Reverse[5]:="b"
ASSIGN Reverse[6]:="g"
ASSIGN Reverse[7]:="f"
DEFINE I:INTEGER
DEFINE Try:INTEGER
ASSIGN Try:=0
ERASE
AT 0410
WRITE Person Name Age Last
LINE 0510;0516
LINE 0518;0524
LINE 0526;0529
LINE 0541;0545
LINE 1441;1446
AT 1440
WRITE First
FOR I:=1,7
  AT 0512+(I*100)
  WRITE "<S,LiNE[ I ]>>
ENDFOR
AT 1510
WRITE Using the letter under the person column, rank these from the LAST name that would appear in a dictionary to the FIRST.

PROMPT ""
ASSIGN I:=1
LOOP I<8
  . MODE INVERSE
  . AT 0543+(I*100)
  . WRITE
  . DO QOne(I,Try,Reverse[I])
  . ERASE 0543+(I*100);0644+(I*100)
  . AT 0543+(I*100)
  . WRITE "<S,Reverse[I]>>
  . ASSIGN I:=(I+1)
ENDLOOP
DO Return
UNIT Five
DEFINE Y[7]:INTEGER
ASSIGN Y[1]:=4
ASSIGN Y[2]:=1
ASSIGN Y[3]:=7
ASSIGN Y[4]:=5
ASSIGN Y[5]:=3
ASSIGN Y[6]:=2
ASSIGN Y[7]:=6
DEFINE Age[7]:INTEGER
DEFINE I:INTEGER
DEFINE J:INTEGER
DEFINE TRY:INTEGER
ASSIGN Line[ 1 ]:="a " Mary d"
ASSIGN Line[ 2 ]:="b " George a"
ASSIGN Line[ 3 ]:="c " Ilsa g"
ASSIGN Line[ 4 ]:="d " Tomas e"
ASSIGN Line[ 5 ]:="e " Tumara c"
ASSIGN Line[ 6 ]:="f " Akbar b"
ASSIGN Line[ 7 ]:="g " Elaine f"
ERASE
FOR I:=1,7
. ASSIGN Age[I]:=0
ENDFOR
AT 0310
WRITE Given a ranking of ages from the youngest to the
oldest, SUPPLY AGES for these people so that the
ranking will be correct.
AT 0710
WRITE Person Name Age Youngest
LINE 0810;0816
LINE 0818;0824
LINE 0826;0830
LINE 0832;0840
FOR I:=1,7
. AT 0812+(I*100)
. WRITE <<S,Line[I]>>
ENDFOR
AT 1732
WRITE Oldest
LINE 1732;1738
ASSIGN TRY:=1
ASSIGN I:=1
ASSIGN QLENGTH:=2
PROMPT "" LOOP (I<7)AND(TRY<4)
. MODE INVERSE
. FOR J:=1,7
. . IF Age[J]=0
109

. . . . AT 0827+(J*100)
. . . . WRITE
. . . . INPUT 0827+(J*100)
. . . . ASSIGN Age[J]:=NUMBER(RESPONSE)
. . . ENDIF
. . . ENDFOR
. . MODE NORMAL
. . LOOP I<7
. . . IF Age[Y[I]]<Age[Y[I+1]]
. . . . ERASE 0827+(Y[I]*100);0929+(Y[I]*100)
. . . . AT 0827+(Y[I]*100)
. . . . WRITE <<S,Age[Y[I]]>> ok
. . . . ASSIGN I:=(I+1)
. . . ELSE
. . . . AT 1910
. . . . WRITE These positions are not ordered correctly.
. . . . FOR J:=I,7
. . . . . . ERASE 0827+(Y[J]*100);0929+(Y[J]*100)
. . . . . . ASSIGN Age[Y[J]]:=0
. . . . . . PAUSE ELAPSED,DELAY
. . . . . . ENDFOR
. . . . ERASE 1910;2053
. . . . OUTLOOP TRUE
. . . . ENDIF
. . . IF I=7
. . . . ERASE 0827+(Y[I]*100);0929+(Y[I]*100)
. . . . AT 0827+(Y[I]*100)
. . . . WRITE <<S,Age[Y[I]]>> ok
. . . ENDIF
. . ENDLOOP
. . ASSIGN TRY:=(TRY+1)
. . IF (TRY<>4) and(I<>7)
. . . AT 1910
. . . WRITE Please try again.
. . . PAUSE ELAPSED,DELAY
. . . ERASE 1910;2027
. . ENDIF
ENDLOOP
ASSIGN QLENGTH:=0
IF I<>7
. . AT 1910
. . WRITE Here is a sample answer
. . FOR J:=1,7
. . . ERASE 0827+(Y[J]*100);0932+(Y[J]*100)
. . . AT 0827+(Y[J]*100)
. . . WRITE <<S,J*10>>
. . . ENDFOR
ENDIF
DO Return
Using the Person (or position) notation instead of the actual values is called indirect access or indirection.

The letters (or numbers) are easier to use because they are unique, in a range from a to g (or 1 to 7).

With the use of indirection the list of items can stay in its original order, yet we now have a means to order the list by some component of an item such as age or name by using an array (or list) of positions (or subscripts).

The rankings we earlier supplied for youngest to oldest and reverse alphabetical would be stored in an array. This array of subscripts or positions would provide a "map" so that the ordering of people would remain unchanged.

Let us look at an example and get familiarized with the notation used for indirectly accessing arrays.
UNIT DataBox

DEFINE I:INTEGER
ASSIGN Pos[1]:="a"
ASSIGN Pos[2]:="b"
ASSIGN Pos[3]:="c"
ASSIGN Pos[4]:="d"
ASSIGN Pos[5]:="e"
ASSIGN Name[1]:="Fred"
ASSIGN Name[2]:="Inna"
ASSIGN Name[3]:="Boyd"
ASSIGN Name[4]:="Anne"
ASSIGN Name[5]:="Mary"
ASSIGN Age[1]:="30"
ASSIGN Age[2]:="10"
ASSIGN Age[3]:="40"
ASSIGN Age[4]:="80"
ASSIGN Age[5]:="20"
AT 0155
WRITE Position Name Age
FOR I:=2,10,2
AT 0158+(I*100)
WRITE "<S,Pos[I/2]> <S,Name[I/2]> <S,Age[I/2]>
ENDFOR
BOX 0054;1379:3
AT 1365
WRITE DATA

UNIT Out( B, VL, SL, C, VR, SR )
DEFINE B:INTEGER
DEFINE VL:INTEGER
DEFINE SL:STRING
DEFINE C:INTEGER
DEFINE VR:INTEGER
DEFINE SR:STRING
DEFINE F:INTEGER
DEFINE L:INTEGER
; ERASE COLUMN
TEST B
VALUE 1
. ERASE 0212+(VL*200);0313+(VL*200)
. AT 0212+(VL*200)
VALUE 2
. ERASE 0230+(VL*200);0331+(VL*200)
. AT 0230+(VL*200)
ENDTEST
WRITE "<S,SL>>
; ERASE FROM DATA BOX
TEST C
VALUE 1
ASSIGN F:=0
ASSIGN L:=1
VALUE 2
ASSIGN F:=7
ASSIGN L:=11
VALUE 3
ASSIGN F:=15
ASSIGN L:=17
ENDTEST
ERASE 0158+(VR*200)+F;0258+(VR*200)+L
AT 0158+(VR*200)+F
WRITE <<S,SR>>
UNIT Eight
DEFINE I: INTEGER
DEFINE RankA[5]: INTEGER
DEFINE ColA[5]: STRING
ASSIGN ColA[1] := "d"
ERASE
DO DataBox
AT 0212
WRITE A
LINE 0311;0314
LINE 0311;1411
LINE 0314;1414
LINE 1411;1414
AT 0416
WRITE First
AT 1216
WRITE Last
AT 0305
WRITE RANK
FOR I := 2, 10, 2
  .  AT 0207+(I*100)
  .  WRITE <<S, INT(I/2)>> <<S, ColA[I/2]>>
ENDFOR
AT 1910
WRITE Column A is the dictionary ordering of names found in the box. A[ RANK ] points to (or is the subscript) for the specific person in the box. The value contained in A[ RANK ] is the person’s location in the box.

DO Return
ERASE 1910;2383
AT 1910
WRITE Notice as the cursor highlights each Rank position, the notation that is used Name[ A[ ] ] is filled in by the rank number and the A column value and its corresponding name in DATA are also highlighted.
AT 1660
WRITE Name[ A[ ] ]
FOR I := 1, 5
  .  MODE INVERSE
AT 1669
WRITE <<S, I>>
DO Out(1, I, ColA[I], 2, RankA[I], Name[RankA[I]] )
MODE NORMAL
PAUSE ELAPSED, DELAY
DO Out(1, I, ColA[I], 2, RankA[I], Name[RankA[I]] )
ERASE 1669;1770
ENDFOR
ERASE 1910;2383
AT 1910
WRITE Try placing a rank value (1 through 5) into the indirect notation and note the matching of the indirect array, Column A and its corresponding name in DATA.

MODE INVERSE
AT 1669
WRITE ASSIGN QLENGTH:=1
PROMPT ""
INPUT 1669
TEST RESPONSE
VALUE "1", "2", "3", "4", "5"
ASSIGN I:=NUMBER(RESPONSE)
DO Out(1, I, ColA[I], 2, RankA[I], Name[RankA[I]] )
OTHER
MODE NORMAL
ERASE 1910;2383
AT 1910
WRITE That was not an expected Rank value.
Suppose that you had typed 3.
MODE INVERSE
AT 1669
WRITE 3
ASSIGN I:=3
DO Out(1, I, ColA[I], 2, RankA[I], Name[RankA[I]] )
ENDTEST
MODE NORMAL
ASSIGN QLENGTH:=0
DO Return
DO Out(1, I, ColA[I], 2, RankA[I], Name[RankA[I]] )
ERASE 1660;1774
ERASE 1910;2383
UNIT Nine

DEFINE I: INTEGER
DEFINE RankB[5]: INTEGER
DEFINE ColB[5]: STRING
ASSIGN RankB[1] := 2
ASSIGN RankB[3] := 1
ASSIGN ColB[1] := "b"
ASSIGN ColB[2] := "e"
ASSIGN ColB[3] := "a"
ASSIGN ColB[4] := "c"
ASSIGN ColB[5] := "d"

AT 1910
WRITE Let us look at another way of organizing DATA.
PAUSE ELAPSED, DELAY
AT 0230
WRITE B
LINE 0329; 0332
LINE 0329; 1429
LINE 0332; 1432
LINE 1429; 1432
AT 0434
WRITE Youngest
AT 1234
WRITE Oldest
FOR I := 2, 10, 2
  . AT 0230 + (I * 100)
  . WRITE "<<S, ColB[I/2]>>"
ENDFOR
ERASE 1910; 2383
AT 1910
WRITE Notice as the cursor highlights each Rank position, the notation that is used Age[ B[ ] ] is filled in by the rank number and the B column value and its corresponding age in DATA are also highlighted.
AT 1660
WRITE Age[ B[ ] ]
FOR I := 1, 5
  . MODE INVERSE
  . AT 1669
  . WRITE "<<S, I>>"
  . DO Out(2, I, ColB[I], 3, RankB[I], Age[RankB[I]] )
  . MODE NORMAL
  . PAUSE ELAPSED, DELAY
  . DO Out(2, I, ColB[I], 3, RankB[I], Age[RankB[I]] )
  . ERASE 1669; 1770
ENDFOR
Try placing a rank value (1 through 5) into the indirect notation and note the matching of the indirect array, Column B and its corresponding age in DATA.

That was not an expected Rank value. Suppose that you had typed 4.
UNIT Ten

DEFINE I:INTEGER
DEFINE RankA[5]:INTEGER
DEFINE ColA[5]:STRING
ASSIGN RankA[1]:=3
ASSIGN RankA[2]:=4
ASSIGN RankA[3]:=2
ASSIGN RankA[4]:=1
ASSIGN RankA[5]:=5
ASSIGN ColA[1]:="d"
ASSIGN ColA[2]:="c"
ASSIGN ColA[3]:="a"
ASSIGN ColA[4]:="b"
ASSIGN ColA[5]:="e"
DEFINE RankB[5]:INTEGER
DEFINE ColB[5]:STRING
ASSIGN RankB[1]:=3
ASSIGN RankB[2]:=1
ASSIGN RankB[3]:=4
ASSIGN RankB[4]:=5
ASSIGN RankB[5]:=2
ASSIGN ColB[1]:="b"
ASSIGN ColB[2]:="e"
ASSIGN ColB[3]:="a"
ASSIGN ColB[4]:="c"
ASSIGN ColB[5]:="d"

AT 1560
WRITE Age[ B[    ] ]
AT 1760
WRITE Name[ A[    ] ]
AT 1910
WRITE Notice as each row of DATA is highlighted the proper values are placed in the notation for Age and Name along with the values in the A and B columns being highlighted.

DO Return
FOR I:=1,5
. MODE INVERSE
. DO Out( 1, RankA[I], ColA[RankA[I]], 2, I, Name[I] )
. DO Out( 2, RankB[I], ColB[RankB[I]], 3, I, Age[I] )
. AT 1769
. WRITE "<S,RankA[I]>>"
. AT 1569
. WRITE "<S,RankB[I]>>"
. PAUSE ELAPSED,DELAY
. MODE NORMAL
. DO Out( 1, RankA[I], ColA[RankA[I]], 2, I, Name[I] )
. DO Out( 2, RankB[I], ColB[RankB[I]], 3, I, Age[I] )
. ERASE 1569;1670
ERASE 1769;1870
ENDFOR
ERASE 1910;2383
DO Return
UNIT Ordering( C, A, Col, R ) $$ Used in 11
DEFINE C:INTEGER $$CHOICE OF COLUMNS
DEFINE A[?] : STRING $$ARRAY
DEFINE Col[?] : STRING $$INDIRECT INDEX
DEFINE R[?] : INTEGER $$RANKING
DEFINE I : INTEGER
DEFINE BLANK : STRING
DEFINE J : INTEGER
DEFINE TRY : INTEGER
ASSIGN TRY := 1

IF C = 2
  . ASSIGN BLANK := " "
  . ASSIGN QLENGTH := 4
ELSE
  . ASSIGN BLANK := " "
  . ASSIGN QLENGTH := 2
ENDIF
FOR I := 1, 5
  . ASSIGN A[I] := ""
ENDFOR
PROMPT ""
ASSIGN J := 1
LOOP (J < 5) AND (TRY < 3)
  . FOR I := 1, 5
    . . IF A[R[I]] = ""
    . . . MODE INVERSE
    . . . DO Out( C-1, I, Col[I], C, R[I], BLANK)
    . . . IF C = 2
    . . . . AT 1769
    . . . ELSE
    . . . . AT 1569
    . . . ENDIF
    . . . ENDIF
    . . . WRITE <<S, I>>
    . . . IF C = 2
    . . . . INPUT 0165+(R[I]*200)
    . . . ELSE
    . . . . INPUT 0173+(R[I]*200)
    . . . ENDIF
    . . . ASSIGN A[R[I]] := RESPONSE
    . . . MODE NORMAL
    . . . DO Out( C-1, I, Col[I], C, R[I], A[R[I]] )
  . . ENDIF
  . ENDIF
  . ENDIF
  LOOP J < 5
    . IF UPPER( A[R[J]]) < (A[R[J+1]])
    . . ASSIGN J := J + 1
    . ELSE
    . . ERASE 1910; 2383
    . . AT 1910
WRITE These positions are not ordered correctly.

FOR I:=J,5
  DO Out( C-1, I, Col[I], C, R[I], BLANK )
  ASSIGN A[R[I]]:="
  PAUSE ELAPSED,DELAY
  ENDIF
  ENDLOOP
  ASSIGN TRY:=TRY+1
  IF (TRY<>3)AND(J<>5)
    ERASE 1910/2383
    AT 1910
    WRITE Please Try Again
    PAUSE ELAPSED,DELAY
    ERASE 1910;2083
  ENDIF
ENDLOOP
UNIT Eleven
DEFINE I: INTEGER
DEFINE RankA[5]: INTEGER
DEFINE ColA[5]: STRING
ASSIGN ColA[1] := "e"
DEFINE RankB[5]: INTEGER
DEFINE ColB[5]: STRING
ASSIGN RankB[1] := 4
ASSIGN RankB[2] := 1
ASSIGN ColB[1] := "d"
ASSIGN ColB[2] := "a"
ASSIGN ColB[3] := "b"
ASSIGN ColB[4] := "e"
ASSIGN ColB[5] := "c"
FOR I:=1,5
  DO Out(1, I, ColA[I], 2, RankA[I], " ")
  DO Out(2, I, ColB[I], 3, RankB[I], " ")
ENDFOR
AT 1910
WRITE The DATA box is empty.
PAUSE ELAPSED, DELAY
ERASE 1910; 2083
AT 1910
WRITE By placing a rank value in the indirect notation to
  to find an Age, say the 4th oldest...
MODE INVERSE
AT 1569
WRITE 4
MODE NORMAL
PAUSE ELAPSED, DELAY
AT 2110
WRITE we can access that spot in the DATA box.
MODE INVERSE
DO Out(2, 4, ColB[4], 3, RankB[4], " ")
MODE NORMAL
DO Return
ERASE 1569; 1670
DO Out( 2, 4, Colb[4], 3, RankB[4]," ")
ERASE 1910;2380
AT 1910
WRITE You are to provide NAMES for the DATA box which follow the ordering found in column A.
DO Return
ERASE 1910;2183
AT 1910
WRITE The names will be entered in the highlighted area in the DATA box.
    If the name has less than 4 letters, press the RETURN, otherwise the name will automatically be entered.
DO Ordering( 2, Name, ColA, RankA )
DO Return
ERASE 1910;2380
AT 1910
WRITE Provide Ages for the DATA box which follow the ordering found under column B. Enter two digit ages.
DO Ordering( 3, Age, ColB, RankB )
DO Return
By using an array of subscripts
(ARRAY[ORDER[RANK]]), any array of data may be
sorted and searched in such a way the data remains
undisturbed.
This fact is especially important when the array has
associated arrays which correspond position by
position.
Instead of moving possibly large amounts of data
multiple times, a simple subscript in an indirect
array can be moved which makes the execution time of
the task much quicker.
UNIT DrawGrid $$ Draws a table
DEFINE I:INTEGER
FOR I:=12,28,4
  LINE 0300+I;1800+I
ENDFOR
FOR I:=300,1800,300
  LINE I+12;I+28
ENDFOR

UNIT FillColW( R, X ) $$ Fills in Left column
DEFINE R:INTEGER $$ROW
DEFINE X:STRING $$VALUE OF W[I]
ERASE (R*300)+114/(R*300)+215
AT (R*300)+114
WRITE <<S,X>>

UNIT FillColXYZ( R, C, X ) $$ Fills in other columns
DEFINE R:INTEGER $$ROW
DEFINE C:INTEGER $$COLUMN 1 2 3
DEFINE X:INTEGER $$VALUE OF X OR Y OR Z
ERASE (R*300)+(C*4)+114;(R*300)+(C*4)+215
AT (R*300)+(C*4)+114
WRITE <<S,X>>

UNIT InitialGrid $$ Labels grid and initializes
DEFINE I:INTEGER
DO DrawGrid
AT 0114
WRITE W X Y Z
AT 0209
WRITE ROW
ASSIGN W[1]:="C"
ASSIGN W[2]:="E"
ASSIGN W[3]:="B"
ASSIGN W[4]:="A"
ASSIGN W[5]:="D"
ASSIGN X[1]:=3
ASSIGN X[2]:=5
ASSIGN X[3]:=2
ASSIGN X[4]:=4
ASSIGN X[5]:=1
ASSIGN Y[1]:=1
ASSIGN Y[2]:=3
ASSIGN Y[3]:=2
ASSIGN Y[4]:=5
ASSIGN Y[5]:=4
ASSIGN Z[1]:=4
ASSIGN Z[2]:=2
ASSIGN Z[3]:=1
ASSIGN Z[4]:=3
ASSIGN Z[5]:=5
FOR I:=1,5
. AT (I*300)+110
. WRITE «S,I>>
. DO FillColW(I, W[I])
. DO FillColXYZ(I, 1, X[I])
. DO FillColXYZ(I, 2, Y[I])
. DO FillColXYZ(I, 3, Z[I])
ENDFOR

UNIT GetChoice( R )
DEFINE R:INTEGER
LOOP TRUE
. TEST RESPONSE
. VALUE "1","2","3","4","5"
. . ASSIGN R:=INT(NUMBER(RESPONSE))
. . ERASE 1935;2183
. . RETURN
. VALUE "[KPE_KEY]"
. . ERASE 1935;2183
. . RETURN
. OTHER
. . ERASE 2013;2114
. . AT 1935
. . WRITE This is not in the range from 1 through 5.
. . Please try again.
. . DO Return
. . INPUT 2013
. ENDTEST
ENDLOOP
UNIT Thirteen
DEFINE R:INTEGER
DEFINE OK:INTEGER
DEFINE LOC:INTEGER
ERASE
DO InitialGrid
PAUSE ELAPSED,DELAY
AT 0535
WRITE Here is a grid which will be used to experiment
with indirect arrays. Column W contains Data and
Columns X, Y and Z are for indirect access to W.
PAUSE ELAPSED,DELAY
AT 1035
WRITE This exercise will deal with 4 increasing levels
of indirect access to W. A level may be repeated
as often as desired. Once 5 correct responses
have been made, pressing the KEYPAD ENTER key
will lead to the next level.
PAUSE ELAPSED,DELAY
AT 1635
WRITE A value in W will be highlighted as the target
value to access. Array notation, such as
W[X[ ]], will be presented and you will be
asked to supply the correct subscript which will
access the targeted value of W.
DO Return
ERASE 0535;2183
AT 0335
SIZE 3
WRITE LEVEL 1
SIZE 1
AT 0535
WRITE At this level we deal with direct access to W.
Below the Grid you will see the array notation.
PAUSE ELAPSED,DELAY
ERASE 2000;2135
AT 2010
WRITE W[
ASSIGN OK:=0
ASSIGN DONE:=FALSE
PROMPT ""
ASSIGN QLENGTH:=1
SEED
LOOP NOT DONE
. AT 0835
. WRITE A value in W is highlighted.
. What subscript (ROW number) provides
access to this value?
. ASSIGN LOC:=INT(RANDOMU(1,6))
MODE INVERSE
DO FillColW( LOC, W[LOC] )
MODE NORMAL
INPUT 2013
DO GetChoice(R)
IF RESPONSE="[KPE_KEY]"
  IF OK>= 5
    ASSIGN DONE:=TRUE
  ENDIF
  DO FillColW( LOC, W[LOC] )
  RETURN
ENDIF
IF LOC=R
  AT 1235
  WRITE Correct!
  ASSIGN OK:=OK+1
  ASSIGN TALLY[1,1]:=TALLY[1,1]+1
  PAUSE ELAPSED,DELAY
ELSE
  ASSIGN TALLY[1,2]:=TALLY[1,2]+1
  MODE INVERSE
  DO FillColW( R, W[R] )
  MODE NORMAL
  AT 1235
  WRITE Sorry, this is incorrect.
       Here is where you have.
  DO Return
  DO FillColW( R, W[R] )
ENDIF
ERASE 1235;1483
ERASE 2013;2114
DO FillColW( LOC, W[LOC] )
IF OK=5
  DO EnterMessage
ENDIF
ERASE 0535;1183
PAUSE ELAPSED,DELAY
ENDLOOP
UNIT Level2( C, W, M, OK )
DEFINE C:INTEGER
DEFINE W[?]:STRING
DEFINE M[?]:INTEGER
DEFINE OK:INTEGER
DEFINE R:INTEGER
DEFINE LOC:INTEGER
PROMPT ""
ASSIGN QLENGTH:=1
SEED
AT 0935
WRITE A value in W is highlighted. What subscript (ROW number) provides access to this value?
ASSIGN LOC:=INT(RANDOMU(1,6))
MODE INVERSE
DO FillColW( M[LOC], W[M[LOC]] )
MODE NORMAL
INPUT 2013
DO GetChoice(R)
IF RESPONSE="[KPE_KEY]"
  IF OK>=5
    . ASSIGN DONE:=TRUE
  ENDIF
  DO FillColW( M[LOC], W[M[LOC]] )
  . ERASE 2000;2383
  . ERASE 0235;1983
  . RETURN
ENDIF
IF R=LOC
  . AT 1335
  . WRITE Correct!
  . ASSIGN OK:=OK+1
  . ASSIGN TALLY[2,1]:=TALLY[2,1]+1
  . PAUSE ELAPSED,DELAY
ELSE
  . ASSIGN TALLY[2,2]:=TALLY[2,2]+1
  . MODE INVERSE
  . DO FillColW( M[R], W[M[R]] )
  . DO FillColXYZ( R, C, M[R])
  . MODE NORMAL
  . AT 1335
  . WRITE Sorry, this is incorrect. Here is where you have.
  . DO Return
  . DO FillColW( M[R], W[M[R]] )
  . DO FillColXYZ( R, C, M[R] )
ENDIF
ERASE 1335;1583
ERASE  2013;2114
DO  FillColW( M[LOC], W[M[LOC]] )
IF  OK=5
  DO  EnterMessage
ENDIF
ERASE  0535;1383
PAUSE  ELAPSED,DELAY
UNIT Fourteen
DEFINE WHICH:INTEGER
DEFINE OK:INTEGER
ERASE
DO InitialGrid
FOR WHICH:=1,5
  DO FillColW( WHICH, W[WHICH] )
ENDFOR
AT 0335
WRITE LEVEL 2
SIZE 3
AT 0535
WRITE At this level we deal with indirect access to W. Below the Grid you will see the array notation.
PAUSE ELAPSED,DELAY
ASSIGN OK:=0
ASSIGN DONE:=FALSE
LOOP NOT DONE
  ERASE 2007;2135
  AT 2007
  ASSIGN WHICH:=INT(RANDOMU(1,4))
  TEST WHICH
  VALUE 1
  . WRITE W[ X[ ] ]
  . DO LEVEL2( 1, W, X, OK )
  VALUE 2
  . WRITE W[ Y[ ] ]
  . DO LEVEL2( 2, W, Y, OK )
  VALUE 3
  . WRITE W[ Z[ ] ]
  . DO LEVEL2( 3, W, Z, OK )
ENDTEST
ENDLOOP
DO Return
UNIT Level3( Cl, C2, W, M, N, OK )
DEFINE Cl:INTEGER
DEFINE C2:INTEGER
DEFINE W[]:STRING
DEFINE M[]:INTEGER
DEFINE N[]:INTEGER
DEFINE OK:INTEGER
DEFINE R:INTEGER
DEFINE LOC:INTEGER
PROMPT ""
ASSIGN QLENGTH:=1
SEED
AT 0935
WRITE A value in W is highlighted. What subscript (ROW number) provides access to this value?
ASSIGN LOC:=INT(RANDOMU(1,6))
MODE INVERSE
DO FillColW( M[N[LOC]], W[M[N[LOC]]] )
MODE NORMAL
INPUT 2013
IF RESPONSE="[KPE_KEY]"
  . IF OK>=5
    . . ASSIGN DONE:=TRUE
    . ELSE
      . . DO FillColW( M[N[LOC]], W[M[N[LOC]]] )
      . . ERASE 2000;2336
      . . ERASE 0235;1983
      . . RETURN
ENDIF
IF R=LOC
  . AT 1435
  . WRITE Correct!
  . ASSIGN OK:=OK+1
  . ASSIGN TALLY[3,1]:=TALLY[3,1]+1
  . PAUSE ELAPSED,DELAY
ELSE
  . ASSIGN TALLY[3,2]:=TALLY[3,2]+1
  . MODE INVERSE
  . DO FillColW( M[N[R]], W[M[N[R]]] )
  . DO FillColXYZ( R, C2, N[R] )
  . DO FillColXYZ( N[R], Cl, M[N[R]] )
  . MODE NORMAL
  . AT 1435
  . WRITE Sorry, this is incorrect. Here is where you have.
  . DO Return
  . DO FillColW( M[N[R]], W[M[N[R]]] )
DO FillColXYZ( R, C2, N[R] )
  . DO FillColXYZ( N[R], C1, M[N[R]] )
ENDIF
ERASE 1435;1683
ERASE 2013;2114
DO FillColW( M[N[LOC]], W[M[N[LOC]]] )
IF OK=5
  . DO EnterMessage
ENDIF
ERASE 0535;1483
PAUSE ELAPSED,DELAY
UNIT Fifteen
DEFINE WHICH:INTEGER
DEFINE OK:INTEGER
ERASE
DO InitialGrid
FOR WHICH:=1,5
  . DO FillColW( WHICH, W[WHICH] )
  . DO FillColXYZ( WHICH, 1, X[WHICH] )
  . DO FillColXYZ( WHICH, 2, Y[WHICH] )
  . DO FillColXYZ( WHICH, 3, Z[WHICH] )
ENDFOR
AT 0335
SIZE 3
WRITE LEVEL 3
SIZE 1
AT 0535
WRITE At this level we deal with double indirect
access to W. Below the Grid you will see
the array notation.
PAUSE ELAPSED,DELAY
ASSIGN OK:=0
ASSIGN DONE:=FALSE
LOOP NOT DONE
  . ERASE 2004;2135
  . AT 2004
  . ASSIGN WHICH:=INT(RANDOMU(1,7))
  . TEST WHICH
  . VALUE 1
    . WRITE W[ X[ Y[ ] ] ]
    . DO LEVEL3( 1, 2, W, X, Y, OK )
  . VALUE 2
    . WRITE W[ Y[ X[ ] ] ]
    . DO LEVEL3( 2, 1, W, Y, X, OK )
  . VALUE 3
    . WRITE W[ Y[ Z[ ] ] ]
    . DO LEVEL3( 2, 3, W, Y, Z, OK )
  . VALUE 4
    . WRITE W[ Z[ Y[ ] ] ]
    . DO LEVEL3( 3, 2, W, Z, Y, OK )
  . VALUE 5
    . WRITE W[ Z[ X[ ] ] ]
    . DO LEVEL3( 1, 3, W, Z, X, OK )
  . VALUE 6
    . WRITE W[ X[ Z[ ] ] ]
    . DO LEVEL3( 3, 1, W, Z, X, OK )
  . ENDT
ENDLOOP
ERASE 2000;2383
DO Return
 UNIT LEVEL4( C1, C2, C3, L, M, N, OK )
 DEFINE C1: INTEGER
 DEFINE C2: INTEGER
 DEFINE C3: INTEGER
 DEFINE L[?]: INTEGER
 DEFINE M[?]: INTEGER
 DEFINE N[?]: INTEGER
 DEFINE OK: INTEGER
 DEFINE R: INTEGER
 DEFINE LOC: INTEGER
 PROMPT ""
 ASSIGN QLENGTH:=1

 SEED
 AT 1235
 WRITE What subscript (ROW number) provides access to this value?
 ASSIGN LOC:=INT(RANDOMU(1, 6))

 MODE INVERSE
 DO FillColXYZ( M[N[LOC]], C1, L[M[N[LOC]]] )
 MODE NORMAL
 INPUT 2013
 DO GetChoice(R)
 IF RESPONSE="[KPE_KEY]"
   . IF OK>=5
   . . ASSIGN DONE:=TRUE
   . ENDIF
   . DO FillColXYZ( M[N[LOC]], C1, L[M[N[LOC]]] )
   . ERASE 2000; 2336
   . ERASE 0235; 1983
   . RETURN
 ENDIF
 IF R=LOC
   . AT 1635
   . WRITE Correct!
   . ASSIGN OK:=OK+1
   . ASSIGN TALLY[4,1]:=TALLY[4,1]+1
   . PAUSE ELAPSED, DELAY
 ELSE
   . ASSIGN TALLY[4,2]:=TALLY[4,2]+1
   . MODE INVERSE
   . DO FillColXYZ( M[N[R]], C1, L[M[N[R]]] )
   . DO FillColXYZ( R, C3, N[R] )
   . DO FillColXYZ( N[R], C2, M[N[R]] )
   . MODE NORMAL
   . AT 1635
   . WRITE Sorry, this is incorrect. Here is where you have.
   . DO Return
   . DO FillColXYZ( M[N[R]], C1, L[M[N[R]]] )
   .
.  DO  FillColXYZ( R, C3, N[R] )
.  DO  FillColXYZ( N[R], C2, M[N[R]] )
ENDIF
ERASE  1235;1883
ERASE  2013;2114
DO  FillColXYZ( M[N[LOC]], C1, L[M[N[LOC]]] )
IF  OK=5
.  DO  EnterMessage
ENDIF
ERASE  0535;1283
PAUSE  ELAPSED,DELAY
UNIT Sixteen
DEFINE WHICH:INTEGER
DEFINE OK:INTEGER
ERASE
DO InitialGrid
FOR WHICH:=1,5
  DO FillColW( WHICH, W[WHICH] )
  DO FillColXYZ( WHICH, 1, X[WHICH] )
  DO FillColXYZ( WHICH, 2, Y[WHICH] )
  DO FillColXYZ( WHICH, 3, Z[WHICH] )
ENDFOR
AT 0335
SIZE 3
WRITE LEVEL 4
SIZE 1
AT 0535
WRITE We can also double indirectly access one of our arrays of subscripts since they are only integer values after all. One of the entries into X, Y or Z will be highlighted and you are to supply the correct subscript through the use of the other two arrays.
PAUSE ELAPSED,DELAY
ASSIGN OK:=0
ASSIGN DONE:=FALSE
LOOP NOT DONE
  ERASE 2004;2135
  AT 2004
  ASSIGN WHICH:=INT(RANDMU(1,7))
  TEST WHICH
  VALUE 1
    . WRITE Z[ X[ Y[ ] ] ]
    . DO LEVEL4( 3, 1, 2, Z, X, Y, OK )
  VALUE 2
    . WRITE Z[ Y[ X[ ] ] ]
    . DO LEVEL4( 3, 2, 1, Z, Y, X, OK )
  VALUE 3
    . WRITE X[ Y[ Z[ ] ] ]
    . DO LEVEL4( 1, 2, 3, X, Y, Z, OK )
  VALUE 4
    . WRITE X[ Z[ Y[ ] ] ]
    . DO LEVEL4( 1, 3, 2, X, Z, Y, OK )
  VALUE 5
    . WRITE Y[ X[ Z[ ] ] ]
    . DO LEVEL4( 2, 1, 3, Y, X, Z, OK )
  VALUE 6
    . WRITE Y[ Z[ X[ ] ] ]
    . DO LEVEL4( 2, 3, 1, Y, Z, X, OK )
ENDTEST
ENDLOOP
ERASE 2000;2383
DO    Return
UNIT Menu
ERASE
ASSIGN QLENGTH:=0
AT 0510
WRITE Would you like to go back to one of the levels and practice it again? Yes or No
INPUT 0630
TEST RESPONSE
VALUE "No","N","n","no","NO"
AT 1210
WRITE Thank you for using this program.
DO Return
OTHER
AT 0710
WRITE Which level? 1, 2, 3 or 4?
LOOP TRUE
. . INPUT 0737
. . TEST RESPONSE
. . VALUE "1"
. . . . DO Thirteen
. . . . OUTLOOP TRUE
. . . . VALUE "2"
. . . . DO Fourteen
. . . . OUTLOOP TRUE
. . . . VALUE "3"
. . . . DO Fifteen
. . . . OUTLOOP TRUE
. . . . VALUE "4"
. . . . DO Sixteen
. . . . OUTLOOP TRUE
. . OTHER
. . . AT 0910
. . . . WRITE This is not in the range from 1 through 4. Please try again.
. . . . DO Return
. . . . ERASE 0910;1083
. . . . ERASE 0737;0838
. . . . INPUT 0737
. . . ENDTEST
. . ENDLOOP
. ERASE
AT 0510
WRITE Would you like to try another?
AT 0620
WRITE Yes or No?
ENDTEST
UNIT Seventeen
DEFINE I:INTEGER
ERASE
AT 0510
WRITE Here are the results of the indirection exercise with W, X, Y and Z.
SIZE 2
FOR I:=1,4
   AT 0510+(I*300)
   WRITE LEVEL <<S,I>> CORRECT <<S,TALLY[I,1]>>
   INCORRECT <<S,TALLY[I,2]>>
ENDFOR
SIZE 1
DO Return
ERASE
SIZE 5
AT 0310
WRITE FINISH OF INDUCTION WITH ARRAYS
SIZE 1
PAUSE ELAPSED,DELAY
ENDLESSON
LESSON Control
DEFINE DELAY : REAL
ASSIGN DELAY := 2.0
CCOLOR ALL
FCOLOR GREEN,1
OPEN "Contol.dat", 4, WRITE, SEQUENTIAL
DO Zero
CLOSE 4
DO One
DO Two
DO Three
DO Four
DO Five
DO Six
DO Seven
DO Eight
DO Nine
DO Ten
DO Eleven
DO Twelve
DO Thirteen
DO Fourteen
UNIT Return
AT 2255
WRITE press RETURN to continue
BOX 2261;2369
PAUSE
ERASE 2155;2383

UNIT Zero
SIZE 4
AT 100,50
WRITE Welcome
to
Indirection
SIZE 1
DO Return
ERASE
AT 0515
WRITE Please enter your student number.
INPUT *
PUT 4,RESPONSE
ERASE
UNIT One
AT 0515
WRITE Suppose we have 100 salaries
with corresponding social security numbers
stored in two arrays, SALARY and SOCSEC.
PAUSE ELAPSED,DELAY
AT 1015
WRITE If we wish to find the highest salary,
we could write the code which would
find that value ...
LINE 1138;1145
DO Return
ERASE

UNIT Two
AT 0520
WRITE Highest := SALARY[ 1 ];
   FOR I := 2 TO 100 DO
      IF SALARY[ I ] > Highest THEN
         Highest := SALARY[ I ];
      Writeln('Highest salary is $',Highest:7:2)
   DO Return
ERASE 2100;2383
UNIT Three
AT 1610
WRITE   NOTE: the highest salary will be in variable
Highest.
MODE REPLACE
MODE INVERSE
ERASE   0822;0929
AT 0822
WRITE   Highest
MODE REPLACE
MODE NORMAL
PAUSE ELAPSED,DELAY
AT 1810
WRITE   We do NOT know which of the 100 salaries was the
highest, we only know the amount.
LINE 2037;2043
DO Return
ERASE  1510;2083
AT 1610
WRITE   If we wished to print the related social security number
we would be out of luck. We have no way of finding that
number except by keeping another variable to hold the
social security number of the highest salary we found.
DO Return
ERASE   0800;1083
MODE REPLACE
MODE INVERSE
AT 0420
WRITE   SSNumber := SOCSEC[ I ];
MODE REPLACE
MODE NORMAL
AT 0820
WRITE BEGIN
   Highest := SALARY[ I ];
AT 1020
MODE REPLACE
MODE INVERSE
WRITE SSNumber := SOCSEC[ I ]
WRITE MODE REPLACE
MODE NORMAL
AT 1120
WRITE END;
   Writeln('Highest salary is $', Highest:7:2 );
MODE REPLACE
MODE INVERSE
AT 1220
WRITE Writeln('The corresponding social security number',
   SSNumber:9 );
MODE REPLACE
MODE NORMAL
DO Return
ERASE
An alternative approach to solving the problem is to have one variable keep track of the LOCATION of the highest salary.

This one LOCATION variable will have a value which will be the subscript (or index) for SALARY!

We can also use this variable as the index into the SOCSEC array since both arrays positions correspond.

The solution is now modified to incorporate the LOCATION variable ...
UNIT Five
AT 0520
WRITE LOCATION := 1;
FOR I := 2 TO 100 DO
    IF SALARY[I] > SALARY[LOCATION] THEN
        LOCATION := I;
    Writeln('Highest salary is $', SALARY[LOCATION]:7:2);
    Writeln('The corresponding social security number is ',
        SOCSEC[LOCATION]:9);
DO Return
AT 1320
WRITE Instead of ...
AT 1520
WRITE Highest := SALARY[I];
FOR I := 2 TO 100 DO
    IF SALARY[I] > Highest THEN
        Highest := SALARY[I];
    Writeln('Highest salary is $', Highest:7:2)
DO Return
The social security number which is associated with the highest salary is obtained by \texttt{SOCSEC[ LOCATION]}. The variable \texttt{LOCATION} indirectly allows us to obtain values in the two arrays. This notion of indirection can be expanded to deal with the entire array.
UNIT Seven
AT 0315
WRITE Suppose we wish to have the pair of arrays ordered from HIGHEST to LOWEST salary but also retain their original ordering.

LINE 0620;0641
PAUSE ELAPSED,DELAY
AT 0715
WRITE One way we could perform this task is to duplicate both arrays and sort the duplicates.
PAUSE ELAPSED,DELAY
AT 1015
WRITE If we had more than just two arrays of data, such as names, ages, employers and so on, we could have many different potential orderings which would require a great many duplicate arrays.
PAUSE ELAPSED,DELAY
AT 1515
WRITE In other words, we would need a copy of every array for each possible ordering. An item having 3 fields such as name, age and salary would require 6 duplicates for each array if we wished only an ascending and descending ranking.
AT 2115
WRITE One hundred of such items requires 2100 array elements!
DO Return
ERASE
With the concept of indirection by using an index or location into the arrays, we could create an array of indices or LOCATIONS which would allow us to keep the arrays in their original order. This LOCATIONS array would provide us with a "map" of how the arrays should be ordered. Only one array would be required to order ALL data arrays in one specified order. One hundred items of name, age and salary ranked in ascending and descending order would require only 900 array elements instead of the previous 2100!
UNIT Nine
AT 0515
WRITE For example, if we had these pairs:
PAUSE ELAPSED,DELAY
AT 0740
WRITE SALARY   SOCSEC
19,901  498-39-9032
15,392  312-02-0017
12,539  382-77-8348
25,836  465-29-5884
18,937  591-48-2201
LINE 0840;0846
LINE 0851;0857
PAUSE ELAPSED,DELAY
AT 1415
WRITE Location would be the "map" which tells us
PAUSE ELAPSED,DELAY
AT 0715
WRITE Location Subscript
  4  1
  1  2
  5  3
  2  4
  3  5
LINE 0815;0823
LINE 0825;0834
PAUSE ELAPSED,DELAY
AT 1515
WRITE that SALARY[4] is the highest,
MODE REPLACE
MODE inverse
AT 1140
WRITE 25,836
PAUSE ELAPSED,DELAY
MODE REPLACE
MODE NORMAL
ERASE 1140;1246
AT 1140
WRITE 25,836
AT 1615
WRITE followed by SALARY[1]
MODE REPLACE
MODE inverse
AT 0840
WRITE 19,901
PAUSE ELAPSED,DELAY
MODE REPLACE
MODE NORMAL
ERASE 0840;0946
AT 0840
WRITE 19,901
LINE 0840;0846
AT 1715
WRITE and so on.
PAUSE ELAPSED,DELAY
AT 1915
WRITE In effect, we are expanding a single variable, LOCATION, into an array LOCATION.
DO Return
UNIT Ten
ERASE
AT 0515
WRITE Initially, the subscript of each element of LOCATION would be the same as its value.
PAUSE ELAPSED,DELAY
AT 0815
WRITE These values would then be switched around dependent on how we wished to organize the data that LOCATION indirectly accesses.
PAUSE ELAPSED,DELAY
AT 1215
WRITE If we wish to sort the data in descending order by salary, we could include the code we used earlier to create a selection sort which would require a variable, HIGH, to find the position of the highest salary.
DO Return

UNIT Eleven
ERASE
AT 0220
WRITE FOR I := 1 TO 100 DO
LOCATION[ I ] := I;
FOR I := 1 TO 99 DO
BEGIN
HIGH := I;
FOR J := I + 1 TO 100 DO
HIGH := J;
TEMP := LOCATION[ I ];
LOCATION[ I ] := LOCATION[ HIGH ];
LOCATION[ HIGH ] := TEMP
END;
DO Return
UNIT Twelve
AT 1515
WRITE Instead of directly accessing SALARY as we did earlier with SALARY[ LOCATION ], we use LOCATION[ X ] as the subscript to find which SALARY is temporarily in the Xth position during the sort.

PAUSE ELAPSED,DELAY
AT 2015
WRITE Tracing through this code with the example data we can see how the values (subscripts) in LOCATION are rearranged.

DO Return
UNIT Thirteen
ERASE 1515;2383
ERASE 0234;0337
AT 0234
WRITE 5
ERASE 0434;0536
AT 0434
WRITE 4
ERASE 0742;0845
AT 0742
WRITE 5
AT 1515
WRITE LOCATION Subscript SALARY SOCSEC
  1 19,901 498-39-9032
  2 15,392 312-02-0017
  3 12,539 382-77-8348
  4 25,836 465-29-5884
  5 18,937 591-48-2201

LINE 1615;1623
LINE 1625;1634
LINE 1636;1642
LINE 1647;1653
PAUSE ELAPSED,DELAY
AT 1618
WRITE 1
  2
  3
  4
  5
PAUSE ELAPSED,DELAY
DEFINE STRSAL[5]:STRING
DEFINE SALARY[5]:INTEGER
DEFINE LOC[5]:INTEGER
DEFINE ISTR:STRING
DEFINE JSTR:STRING
DEFINE I:INTEGER
DEFINE J:INTEGER
DEFINE HI:INTEGER
DEFINE TEMP:INTEGER
ASSIGN STRSAL[1]:="19,901"
ASSIGN STRSAL[2]:="15,392"
ASSIGN STRSAL[3]:="12,539"
ASSIGN STRSAL[4]:="25,836"
ASSIGN STRSAL[5]:="18,937"
ASSIGN SALARY[1]:=19901
ASSIGN SALARY[2]:=15392
ASSIGN SALARY[3]:=12539
ASSIGN SALARY[4]:=25836
ASSIGN SALARY[5]:=18937
ASSIGN ISTR:="I->"
ASSIGN JSTR:="<-J"
FOR I:=1,5
  ASSIGN LOC[I]:=I
ENDFOR
AT 1605
WRITE HIGH=
FOR I:=1,4
  ASSIGN HI:=I
  AT 1515+(I*100)
  WRITE <<S,ISTR>>
  DO Hilite( LOC[I], STRSAL[LOC[I]] )
  ERASE 1610;1711
  AT 1610
  WRITE <<S,I>>
  FOR J:=I+1,5
    AT 1519+(J*100)
    WRITE <<S,JSTR>>
    DO Hilite( LOC[J], STRSAL[LOC[J]] )
    PAUSE ELAPSED,DELAY
    IF SALARY[LOC[J]] > SALARY[LOC[HI]]
      ASSIGN HI:=J
      ERASE 1610;1711
      AT 1610
      WRITE <<S,HI>>
      ELSE
      DO Unhilite( LOC[J], STRSAL[LOC[J]] )
      ENDIF
    ERASE 1519+(J*100);1622+(J*100)
  ENDFOR
  DO Unhilite( LOC[HI], STRSAL[LOC[HI]] )
  ASSIGN TEMP:=LOC[HI]
  ASSIGN LOC[HI]:=LOC[I]
  ASSIGN LOC[I]:=TEMP
  ERASE 1515+(I*100);1619+(I*100)
  IF I=1
    LINE 1615;1619
  ENDIF
  AT 1518+(I*100)
  WRITE <<S,LOC[I]]>
  ERASE 1518+(HI*100);1619+(HI*100)
  AT 1518+(HI*100)
  WRITE <<S,LOC[HI]]>
  PAUSE ELAPSED,DELAY
ENDFOR
ERASE 1605;1711
DO Return
In Summary

The use of an array of subscripts, LOCATION, to indirectly access a sequence of arrays allows for great savings of storage and reduces the amount of data that must be copied.

It also provides a means for many possible organizations for the same data while leaving the original data alone.

END OF INDIRECTION
APPENDIX C

LECTURE MATERIAL
FREQUENCIES OF QUIZ SCORES

Suppose we have 100 quiz scores which range from 0 to 10 stored in an array QUIZ. Recall that to find the arithmetic mean (average) we will sum the quiz scores and divide by the quantity of scores. For example the code would be as follows

\[
\text{Sum} := 0.0; \\
\text{For } I := 1 \text{ to 100 do} \\
\quad \text{Sum} := \text{Sum} + \text{QUIZ}[I]; \\
\text{Mean} := \text{Sum} \div 100.0
\]

Another way that information about the scores could be presented would be to represent the scores visually in the form of a bar graph by the frequency of each score. If we wished to determine the number of scores which were equal to zero then we simply loop through the array and tally each time we find an array element equal to zero.

\[
\text{ZeroTally} := 0; \\
\text{For } I := 1 \text{ to 100 do} \\
\quad \text{If QUIZ}[I] = 0 \text{ Then ZeroTally} := \text{ZeroTally} + 1;
\]

A interpolation line at the beginning of the output could be printed followed by a special symbol such as an asterisk (*) to represent each occurrence of a zero score.

\[
\text{Writeln('} \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 4\text{');} \\
\text{Writeln('Score 0...5...0...5...0...5...0...5...0');} \\
\text{Write(0:5,' ');} \\
\text{For } J := 1 \text{ to ZeroTally do} \\
\quad \text{Write('*');} \\
\text{Writeln}
\]

Since the tallies must be made for each score the use of an array TALLY will allow us to generalize our solution. We can use a For loop for each of the 11 score values and another for each of the 100 scores.

\[
\text{For Value := 0 to 10 do} \\
\quad \text{Begin} \\
\quad \quad \text{TALLY}[\text{Value}] := 0; \\
\quad \quad \text{For } I := 1 \text{ to 100 do} \\
\quad \quad \quad \text{If QUIZ}[I] = \text{Value} \\
\quad \quad \quad \quad \text{Then TALLY}[\text{Value}] := \text{TALLY}[\text{Value}] + 1 \\
\quad \quad \text{End;} \\
\text{Writeln('} \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 4\text{');} \\
\text{Writeln('Score 1...5...0...5...0...5...0...5...0');}
\]
For Value := 0 to 10 do
  Begin
    Write( Value : 5, ' ' );
    For J := 1 to TALLY[ Value ] do
      Write( '*' );
    Writeln
  End.

This example solution can be refined further. Notice that the
If statement will be executed 1100 times because it is inside
the nested for loops.

Instead of looking at the problem as searching the array
for all zeros, then all ones and so on, we could look through
the list once and increment the proper TALLY. How do we know
which is the proper TALLY? The quiz score is our index into
TALLY. This means that the subscript of TALLY will itself be
a subscripted variable. The first portion of the solution
will be modified to be the following:

For Value := 0 to 10 do
  TALLY[ Value ] := 0;
For I := 1 to 100 do

The If statement in the original solution was used to specify
which TALLY was to be incremented. The current solution uses
the each element of QUIZ be the subscript into TALLY. Try
tracing this code with a few scores in QUIZ. Notice that the
number of times the statements are executed in the current
example is approximately 10 times less than our original
solution!

FREQUENCIES OF LETTERS

Suppose we switch from finding frequencies of quiz scores
and found frequencies of letters in character arrays. The
subscripts for the TALLY array would be of type character. If
we wished to count the occurrences of various characters of a
line of input, a program segment could be as follows:

For I := 0 to 255 do
  TALLY[ CHR( I ) ] := 0; {initialize the array}
While Not Eoln do
  Begin
    Read( Ch );
    TALLY[ Ch ] := TALLY[ Ch ] + 1
  End;

If the characters to be tallied were contained in an character
array, say STRING, then we would substitute for the While
statement a For loop that would range from 1 to the length of STRING.

Read( STRING );
For J := 1 to N do
To output the frequencies, we would perform similar statements as we did with the quiz scores,

Writeln(' 1 1 2 2 3 3 4');
Writeln('Char 1...5...0...5...0...5...0...5...0');
For I := 0 to 255 do
    Begin
        Write( CHR( I ) : 4, ' ');
        For J := 1 to TALLY[ CHR( I ) ] do
            Write('*');
        Writeln
    End .

Counting the occurrences of characters can be applied to a interesting problem. Words such as STOP, POTS, TOPS are called anagrams because they contain the same letters but in different orders. If we were given two or more words and asked whether any anagrams were present, the frequencies of letters for anagrams would be identical. For example if we were given the words TERSE, STEER, STREET, RESET and ERNST, we would say that TERSE, STEER and RESET were anagrams because they have two E's, one R, one S and one T whereas STREET has two T's and ERNST has only one E and has one N.

Programming Problem Example

Write a program which will read an undetermined number of words which may contain up to 10 letters and print out all sets of anagrams.

SEARCHING AN ARRAY WITH AN INDEX

As we have seen with the previous frequency problems, there are several ways in which way can access information contained in an array of which one way may be of greater benefit for certain problems. Another example of this idea can be shown by investigating a previously studied problem such as finding the largest value of an array. Suppose we have 100 salaries stored in SALARY and we wish to find the highest. The code for that could be:

Highest := SALARY[ 1 ];
For I := 2 to 100 do
    If SALARY[ I ] > Highest Then Highest := SALARY[ I ];
This solution becomes cumbersome if along with the salaries we have companion arrays which contain names and social security numbers. If we wished for the name and social security number of the person with the highest salary, we have no idea where in the other arrays to find them because our solution did not save the position of the highest salary. Our solution only saved the value of the highest salary. By saving the position we have an index into all the arrays or we might say that we can \textit{indirectly} access the arrays \textit{through the use of the position}. In other words, we do not have the highest salary directly in hand but we can get it through the use of position which tells us where it is in the \texttt{SALARY} array and then gives us the corresponding name and social security number from those arrays. The solution is modified to be

\begin{verbatim}
HighestIndex := 1;
For I := 2 to 100 do
  If Salary[ I ] > Salary[ HighestIndex ]
    Then HighestIndex := I;
Writeln( Name[ HighestIndex], SocialSecNum[ HighestIndex] ).
\end{verbatim}

This concept of \textit{indirectly} accessing the array can be of use in other ways as well. Suppose we have a parking lot in which customers must pay to park according to the amount of time their cars stay on the lot. We will keep track of cars by license plates. As cars arrive we place the license plates into an array,

\begin{verbatim}
I := 1;
While Not Eof do
  Begin
    Read( LicensePlate );
    PARKINGLIST[ I ] := LicensePlate;
    I := I + 1
  End
\end{verbatim}

If we wish to have the license plates ordered for some report, we could sort the \texttt{PARKINGLIST}. However, the original ordering of \texttt{PARKINGLIST} is important because it helps tell us when cars arrived. By sorting \texttt{PARKINGLIST} directly, that initial ordering will be destroyed.

With the idea of indirect accessing we could keep the original ordering of \texttt{PARKINGLIST} the same and have a list of indices which would tell us the sorted order. For instance if we had the license plates:

\texttt{RTE 348, UWE 903, IRY 024, BAK 204, KAA 920}

the ordering will be the fourth, third, fifth, first and second. Represented in a different way
The use of INDEX (which is subscripted) as a subscript into PARKINGLIST has the same notation as the frequency problems, specifically

\[
PARKINGLIST[ INDEX[ I ] ]
\]

Each element of INDEX will initially contain the value of its subscript, so INDEX[ 1 ] will be 1 and so on.

Recall how a selection sort works,

For \( A := 1 \) to \( N - 1 \) do

Begin

\[
\text{StartIndex} := A;
\]

For \( B := A + 1 \) to \( N \) do

If PARKINGLIST[ B ] < PARKINGLIST[ StartIndex ]

Then \( \text{StartIndex} := B; \)

\[
\text{Temp} := PARKINGLIST[ A ];
\]

PARKINGLIST[ A ] := PARKINGLIST[ StartIndex ];

PARKINGLIST[ StartIndex ] := Temp

End .

We modify the solution to include the array INDEX which allows us indirect access to PARKINGLIST. So everywhere there is a subscript we will substitute a reference to INDEX,

For \( I := 1 \) to \( N \) do INDEX[ I ] := I;

For \( A := 1 \) to \( N - 1 \) do

Begin

\[
\text{StartIndex} := \text{INDEX}[ A ];
\]

For \( B := A + 1 \) to \( N \) do

If PARKINGLIST[ INDEX[ B ] ] <

PARKINGLIST[ StartIndex ]

Then \( \text{StartIndex} := \text{INDEX}[ B ]; \)

{ REST OF THE SORT }

Each time the program goes through the inner loop (For B), StartIndex contains the subscript from PARKINGLIST which comes first. The problem we face now is the same one we have faced earlier; we know which subscript of PARKINGLIST has the beginning license plate but we do not know where in INDEX it was. (Note that initially we do know because the subscripts and values of INDEX are the same but as sorting progresses, those two items will vary.) We need to keep the position into INDEX which we will call IndirectIndex. We will then swap PARKINGLIST subscripts instead of the license plates,
For I := 1 to N do
  INDEX[ I ] := I;
For A := 1 to N - 1 do
  Begin
    IndirectIndex := A;
    For B := A + 1 to N do
      If PARKINGLIST[ INDEX[ B ] ] < PARKINGLIST[ INDEX[ IndirectIndex ] ]
        Then IndirectIndex := B;
    Temp := INDEX[ A ];
    INDEX[ A ] := INDEX[ IndirectIndex ];
    INDEX[ IndirectIndex ] := Temp
  End

If we used a bubble sort instead of a selection sort the variable IndirectIndex would not be required. In this sort neighboring values of INDEX are compared. If the values in PARKINGLIST to which they point are not in order the values of INDEX are switched.

For I := 1 to N do
  INDEX[ I ] := I;
For A := 1 to N - 1 do
  For B := N downto A + 1 do
      Begin
        Temp := INDEX[ B ];
        INDEx[ B ] := INDEX[ B - 1 ];
        INDEX[ B - 1 ] := Temp
      End

Using these two sorting examples, let us trace through them to satisfy ourselves that they work properly. If we had companion arrays such as owner name and time, our INDEX array could be used to find those corresponding values as well (just like we did in the earlier example).

Programming Problem Example

Using this example and given the time when the car entered and left the lot, find the elapsed time each car was in the lot. Output the elapsed time for each car according to the order the cars arrived and according to the license plate ordering.
APPENDIX D.

COMPUTER-BASED EVALUATION
LESSON Evaluate
DEFINE SYM[5]:STRING
DEFINE W[5]:STRING
DEFINE X[5]:INTEGER
DEFINE Y[5]:INTEGER
DEFINE Z[5]:INTEGER
DEFINE DELAY:REAL
DEFINE TALLY[4,2]:INTEGER
DEFINE ARRD[4]:INTEGER
DEFINE ARR1[4]:INTEGER
DEFINE ARR1L[4]:INTEGER
DEFINE ARR2W[4]:INTEGER
DEFINE ARR2WL[4]:INTEGER
DEFINE ARR2X[4]:INTEGER
DEFINE ARR2XL[4]:INTEGER
DEFINE HOWMANY:INTEGER
DEFINE DIGITS:BOOLEAN
DEFINE NUMSTRING:STRING
CCOLOR ALL
FCOLOR GREEN, 1
ASSIGN HOWMANY:=16
ASSIGN DELAY:=1.0
ASSIGN SYM[1]:="@"
ASSIGN SYM[2]:="&"
ASSIGN SYM[3]:="#"
ASSIGN SYM[4]:="+
ASSIGN SYM[5]:="+
;LOG FULL
SPECS PUNC
SPECS EXACT
DO Intro
DO InitialArrays
OPEN NUMSTRING, 4, WRITE, SEQUENTIAL
ASSIGN DIGITS:=TRUE
DO SymTest
ASSIGN DIGITS:=FALSE
DO SymTest
CLOSE 4
DO Finish
UNIT Return
AT 2255
WRITE press RETURN to continue
BOX 2261;2367
PAUSE
ERASE 2255;2383

UNIT Unhilite( I, Str )
DEFINE Str:STRING
DEFINE I:INTEGER
MODE REPLACE
MODE NORMAL
ERASE 0512+(I*100);0628+(I*100)
AT 0512+(I*100)
WRITE «<<S,Str>>

UNIT Hilite( I, Str )
DEFINE Str:STRING
DEFINE I:INTEGER
MODE REPLACE
MODE INVERSE
ERASE 0512+(I*100);0628+(I*100)
AT 0512+(I*100)
WRITE «<<S,Str>>
MODE REPLACE
MODE NORMAL
This lesson is an evaluation of the topic of indirect access to arrays which was presented in class.

Please enter your student ID number:

You will be presented with a table containing values and subscripts. One element within the table will highlighted which is to be accessed.

Below the table will be an array notation in which you must supply the correct subscript so that the highlighted element can be accessed.

For example...

With this notation...

The correct answer is 4.
Your answers and the time taken to provide an answer will be recorded so that group comparisons can be made.

Now we begin the evaluation!
UNIT InitialArrays
ASSIGN ARRD[1] := 5
ASSIGN ARRD[2] := 2
ASSIGN ARRD[3] := 1
ASSIGN ARRD[4] := 4
ASSIGN ARR1[1] := 3
ASSIGN ARR1[2] := 1
ASSIGN ARR1[3] := 2
ASSIGN ARR1[4] := 2
ASSIGN ARR1L[1] := 4
ASSIGN ARR1L[2] := 5
ASSIGN ARR1L[3] := 2
ASSIGN ARR1L[4] := 3
ASSIGN ARR2W[1] := 1
ASSIGN ARR2W[3] := 3
ASSIGN ARR2W[4] := 2
ASSIGN ARR2WL[1] := 4
ASSIGN ARR2WL[2] := 1
ASSIGN ARR2X[1] := 3
ASSIGN ARR2X[3] := 1
ASSIGN ARR2X[4] := 4
ASSIGN ARR2XL[1] := 5
ASSIGN ARR2XL[2] := 2
ASSIGN ARR2XL[3] := 3
ASSIGN ARR2XL[4] := 1
UNIT SampleGrid
DO DrawGrid
AT 0114
WRITE W X Y Z
AT 0209
WRITE ROW
AT 0410
WRITE 1 D 3 5 2
AT 0710
WRITE 2 C 1 2 4
AT 1010
WRITE 3 4 1 5
MODE REPLACE
MODE INVERSE
AT 1014
WRITE A
MODE OVERLAY
MODE NORMAL
AT 1310
WRITE 4 B 5 3 1
AT 1610
WRITE 5 E 2 4 3

UNIT DrawPath
AT 1311
WRITE ———— ————>
PAUSE ELAPSED,DELAY
LINE 1322;1111
PAUSE ELAPSED,DELAY
AT 1011
WRITE -->
UNIT SymTest
DEFINE I:INTEGER
DEFINE J:INTEGER
DEFINE K:INTEGER
DEFINE MESS:STRING
ERASE
DO Shuffle
DO InitialGrid
SEED
IF DIGITS
   ASSIGN MESS:="DIGITS EVALUATION"
   PUT 4,MESS
ELSE
   ASSIGN MESS:="SYMBOLS EVALUATION"
   PUT 4,MESS
ENDIF
FOR J:=1,4
   FOR K:=1,4
      TEST J
      VALUE 1
      . . DO Direct(K)
      . VALUE 2
      . . DO Indir1(K)
      . VALUE 3
      . . DO Indir2W(K)
      . VALUE 4
      . . DO Indir2XYZ(K)
      . ENDTEST
      ASSIGN MESS:="LATENCY"
      NOW GET LATENCY AFTER EACH ATTEMPT
      . PUT 4,MESS
      . PUT 4,STRING(LATENCY)
      . ASSIGN MESS:="*******"
      . PUT 4,MESS
   ENDFOR
   DO Shuffle
   FOR I:=1,5
      ERASE (I*300)+110;(I*300)+211
      AT (I*300)+110
      IF DIGITS
         WRITE <<S,I>>
         . . DO FillColW( I, W[I] )
         . . DO FillColXYZ( I, 1, STRING(X[I]) )
         . . DO FillColXYZ( I, 2, STRING(Y[I]) )
         . . DO FillColXYZ( I, 3, STRING(Z[I]) )
         . ELSE
            WRITE <<S,SYM[I]>>
            . . DO FillColW( I, W[I] )
            . . DO FillColXYZ( I, 1, SYM[X[I]] )
DO FillColXYZ( I, 2, SYM[Y[I]] )
DO FillColXYZ( I, 3, SYM[Z[I]] )
ENDIF
ENDFOR
ENDFOR
ASSIGN ARR2X[4]:=1 $$ THIS IS AN EXTRA TEST
ADDED FOR WINTER 91-92 CLASS
ASSIGN ARR2XL[4]:=5 $$
DO INDIR2XYZ(4) $$
ASSIGN MESS:="EXTRA TEST LATENCY" $$
PUT 4,MESS $$
PUT 4,STRING(LATENCY) $$
ASSIGN MESS:="*******" $$
PUT 4,MESS $$
ASSIGN ARR2X[4]:=4 $$ NEED TO RESTORE FOR
SYMBOL TEST
ASSIGN ARR2XL[4]:=1 $$
DO Results
UNIT DrawGrid $$ Draws a table
DEFINE I: INTEGER
FOR I:=12,28,4
  LINE 0300+I;1800+I
ENDFOR
FOR I:=300,1800,300
  LINE I+12;I+28
ENDFOR

UNIT FillColW( R, X ) $$ Fills in Left column
DEFINE R: INTEGER $$ROW
DEFINE X: STRING $$VALUE OF W[I]
ERASE (R*300)+114;(R*300)+215
AT (R*300)+114
WRITE <<S,X>>

UNIT FillColXYZ( R, C, X ) $$ Fills in other columns
DEFINE R: INTEGER $$ROW
DEFINE C: INTEGER $$COLUMN 1 2 3
DEFINE X: STRING $$VALUE OF X OR Y OR Z
ERASE (R*300)+(C*4)+114;(R*300)+(C*4)+215
AT (R*300)+(C*4)+114
WRITE <<S,X>>

UNIT Shuffle $$ Special symbols to be shuffled for mapping
DEFINE Temp: STRING
DEFINE I: INTEGER
DEFINE R: INTEGER
FOR I:=1,5
  ASSIGN R:=INT(RANDOMU(1, 6))
  ASSIGN Temp:=SYM[R]
  ASSIGN SYM[R]:=SYM[I]
  ASSIGN SYM[I]:=Temp
ENDFOR

UNIT InitialGrid $$ Labels grid and initializes
DEFINE I: INTEGER
DO DrawGrid
AT 0114
WRITE W X Y Z
AT 0209
WRITE ROW
ASSIGN W[1]:="C"
ASSIGN W[2]:="E"
ASSIGN W[3]:="B"
ASSIGN W[4]:="A"
ASSIGN W[5]:="D"
ASSIGN X[1]:=3
ASSIGN X[2]:=5
ASSIGN X[3]:=2
ASSIGN X[4]:=4
ASSIGN X[5]:=1
ASSIGN Y[1]:=1
ASSIGN Y[2]:=3
ASSIGN Y[3]:=2
ASSIGN Y[4]:=5
ASSIGN Y[5]:=4
ASSIGN Z[1]:=4
ASSIGN Z[2]:=2
ASSIGN Z[3]:=1
ASSIGN Z[4]:=3
ASSIGN Z[5]:=5
FOR I:= 1, 4
  ASSIGN TALLY[I,1]:=0
  ASSIGN TALLY[I,2]:=0
ENDFOR

FOR I:= 1, 5
  ERASE (I*300)+110/(I*300)+211
  AT (I*300)+110
  IF DIGITS
    .  WRITE <<S,I>>
    .  DO FillColW(I, W[I])
    .  DO FillColXYZ(I, 1, STRING(X[I]))
    .  DO FillColXYZ(I, 2, STRING(Y[I]))
    .  DO FillColXYZ(I, 3, STRING(Z[I]))
    ELSE
    .  WRITE <<S,SYM[I]]
    .  DO FillColW(I, W[I])
    .  DO FillColXYZ(I, 1, SYM[X[I]])
    .  DO FillColXYZ(I, 2, SYM[Y[I]])
    .  DO FillColXYZ(I, 3, SYM[Z[I]])
  ENDIF
ENDFOR

UNIT GetChoice( R )
DEFINE R:INTEGER
DEFINE I:INTEGER
IF DIGITS
  DO DigitQ( R )
ELSE
  QUERY 2013
  RIGHT @ | % | & | # | +
  FOR I:= 1, 5
    .  .  IF SYM[I]=RESPONSE
    .  .  .  ASSIGN R:=I
    .  .  ENDIF
  .  ENDFOR
UNIT DigitQ( R )
DEFINE R:INTEGER
QUERY 2013
RIGHT 1|2|3|4|5
    ASSIGN R:=INT(NUMBER(RESPONSE))
WRONG
    AT 1935
    WRITE This is not an integer in the range from 1 through 5
    Please try again.
    DO Return
ENDQ
UNIT Direct(K)
DEFINE K:INTEGER
DEFINE R:INTEGER
DEFINE LOC:INTEGER
PUT 4,CUNIT
ERASE 2000;2135
AT 2010
WRITE W[ ]
PROMPT ""
ASSIGN QLENGTH:=1
SEED
AT 0835
WRITE A value in W is highlighted.
What subscript (ROW symbol) provides access to this value?
ASSIGN LOC:=ARRD[K]
PUT 4,STRING(LOC)
MODE REPLACE
MODE INVERSE
DO FillColW( LOC, W[LOC] )
MODE REPLACE
MODE NORMAL
DO GetChoice(R)
PUT 4,STRING(R)
IF LOC=R
  . ASSIGN TALLY[1,1]:=TALLY[1,1]+1
ELSE
  . ASSIGN TALLY[1,2]:=TALLY[1,2]+1
ENDIF
ERASE 1235;1483
ERASE 2013;2114
DO FillColW( LOC, W[LOC] )
ERASE 0535;1183
PAUSE ELAPSED,DELAY
DO Return
UNIT Level2( C, W, M, K )
DEFINE C:INTEGER
DEFINE W[?]:STRING
DEFINE M[?]:INTEGER
DEFINE K:INTEGER
DEFINE R:INTEGER
DEFINE LOC:INTEGER
PROMPT "" 
ASSIGN QLENGTH:=1
SEED
AT 0935
WRITE A value in W is highlighted.
What subscript (ROW symbol) provides access to this value?
ASSIGN LOC:=ARR1L[K]
PUT 4,STRING(LOC)
MODE REPLACE
MODE INVERSE
DO FillColW( M[LOC], W[M[LOC]] )
MODE REPLACE
MODE NORMAL
DO GetChoice(R)
PUT 4,STRING(R)
IF R=LOC
  . ASSIGN TALLY[2,1]:=TALLY[2,1]+1
ELSE
  . ASSIGN TALLY[2,2]:=TALLY[2,2]+1
ENDIF
ERASE 1335/1583
ERASE 2013/2114
DO FillColW( M[LOC], W[M[LOC]] )
ERASE 0535/1383
PAUSE ELAPSED,DELAY
UNIT Indir1(K)
DEFINE K:INTEGER
DEFINE WHICH:INTEGER
PUT 4,CUNIT
ERASE 2007;2135
AT 2007
ASSIGN WHICH:=ARR1[K]
TEST WHICH
VALUE 1
. WRITE W[ X[] ]
. DO LEVEL2( 1, W, X, K )
VALUE 2
. WRITE W[ Y[] ]
. DO LEVEL2( 2, W, Y, K )
VALUE 3
. WRITE W[ Z[] ]
. DO LEVEL2( 3, W, Z, K )
ENDTEST
DO Return
UNIT Level3( C1, C2, W, M, N, K )
DEFINE C1:INTEGER
DEFINE C2:INTEGER
DEFINE W[?]:STRING
DEFINE M[?]:INTEGER
DEFINE N[?]:INTEGER
DEFINE K:INTEGER
DEFINE R:INTEGER
DEFINE LOG:INTEGER
PROMPT ""
ASSIGN QLENGTH:=1
SEED
AT 0935
WRITE A value in W is highlighted. What subscript (ROW symbol) provides access to this value?
ASSIGN LOC:=ARR2WL[K]
PUT 4,STRING(LOC)
MODE REPLACE
MODE INVERSE
DO FillColW( M[N[LOC]], W[M[N[LOC]]] )
MODE REPLACE
MODE NORMAL
DO GetChoice(R)
PUT 4,STRING(R)
IF R=LOC
  ASSIGN TALLY[3,1]:=TALLY[3,1]+1
ELSE
  ASSIGN TALLY[3,2]:=TALLY[3,2]+1
ENDIF
ERASE 1435/1683
ERASE 2013/2114
DO FillColW( M[N[LOC]], W[M[N[LOC]]] )
ERASE 0535/1483
PAUSE ELAPSED,DELAY
UNIT Indir2W(K)
DEFINE K:INTEGER
DEFINE WHICH:INTEGER
PUT 4,CUNIT
ERASE 2004;2135
AT 2004
ASSIGN WHICH:=ARR2W[K]
TEST WHICH
VALUE 1
. WRITE W[ X[ Y[   ] ] ]
. DO LEVEL3( 1, 2, W, X, Y, K )
VALUE 2
. WRITE W[ Y[ X[   ] ] ]
. DO LEVEL3( 2, 1, W, Y, X, K )
VALUE 3
. WRITE W[ Y[ Z[   ] ] ]
. DO LEVEL3( 2, 3, W, Y, Z, K )
VALUE 4
. WRITE W[ Z[ Y[   ] ] ]
. DO LEVEL3( 3, 2, W, Z, Y, K )
VALUE 5
. WRITE W[ Z[ Z[   ] ] ]
. DO LEVEL3( 3, 1, W, Z, X, K )
VALUE 6
. WRITE W[ X[ Z[   ] ] ]
. DO LEVEL3( 3, 1, W, Z, X, K )
ENDTEST
ERASE 2000;2383
DO Return
UNIT LEVEL4(C1, C2, C3, L, M, N, K)
DEFINE C1: INTEGER
DEFINE C2: INTEGER
DEFINE C3: INTEGER
DEFINE L[?]: INTEGER
DEFINE M[?]: INTEGER
DEFINE N[?]: INTEGER
DEFINE K: INTEGER
DEFINE R: INTEGER
DEFINE LOG: INTEGER
PROMPT ""
ASSIGN QLENGTH:=1
SEED
AT 1235
WRITE What subscript (ROW symbol) provides access to this value?
ASSIGN LOC:=ARR2XL[K]
PUT 4, STRING(LOC)
MODE REPLACE
MODE INVERSE
IF DIGITS
  DO FillColXYZ( M[N[LOC]], C1, STRING(L[M[N[LOC]]]) )
ELSE
  DO FillColXYZ( M[N[LOC]], C1, SYM[L[M[N[LOC]]]] )
ENDIF
MODE REPLACE
MODE NORMAL
DO GetChoice(R)
PUT 4, STRING(R)
IF R=LOC
  ASSIGN TALLY[4,1]:=TALLY[4,1]+1
ELSE
  ASSIGN TALLY[4,2]:=TALLY[4,2]+1
ENDIF
ERASE 1235; 1883
ERASE 2013; 2114
IF DIGITS
  DO FillColXYZ( M[N[LOC]], C1, STRING(L[M[N[LOC]]]) )
ELSE
  DO FillColXYZ( M[N[LOC]], C1, SYM[L[M[N[LOC]]]] )
ENDIF
ERASE 0535; 1283
PAUSE ELAPSED, DELAY
UNIT Indir2XYZ(K)
DEFINE K:INTEGER
DEFINE WHICH:INTEGER
PUT 4,CUNIT
ERASE 2004;2135
AT 2004
ASSIGN WHICH:=ARR2X[K]
TEST WHICH
VALUE 1
  WRITE Z[ X[ Y[   ] ] ]
  DO LEVEL4( 3, 1, 2, Z, X, Y, K )
VALUE 2
  WRITE Z[ Y[ X[   ] ] ]
  DO LEVEL4( 3, 2, 1, Z, Y, X, K )
VALUE 3
  WRITE X[ Y[ Z[   ] ] ]
  DO LEVEL4( 1, 2, 3, X, Y, Z, K )
VALUE 4
  WRITE X[ Z[ Y[   ] ] ]
  DO LEVEL4( 1, 3, 2, X, Z, Y, K )
VALUE 5
  WRITE Y[ X[ Z[   ] ] ]
  DO LEVEL4( 2, 1, 3, Y, X, Z, K )
VALUE 6
  WRITE Y[ Z[ X[   ] ] ]
  DO LEVEL4( 2, 3, 1, Y, Z, X, K )
ENDTEST
ERASE 2000;2383
DO Return
UNIT Results
DEFINE I:INTEGER
DEFINE MESS:STRING
ERASE
AT 0510
WRITE Here are the results of the indirection exercise with W, X, Y and Z.
SIZE 2
FOR I:=1,4
  AT 0510+(I*300)
  WRITE LEVEL <<S,I>> CORRECT <<S,TALLY[I,1]>>
  INCORRECT <<S,TALLY[I,2]>>
  ASSIGN MESS:="LEVEL"
  PUT 4,MESS
  PUT 4,MES-I
  ASSIGN MESS:="CORRECT"
  PUT 4,MESS
  PUT 4,STRING(TALLY[I,1])
  ASSIGN MESS:="INCORRECT"
  PUT 4,MESS
  PUT 4,STRING(TALLY[I,2])
ENDFOR
SIZE 1
DO Return
ERASE

UNIT Finish
SIZE 5
AT 0310
WRITE FINISH OF EVALUATION WITH ARRAYS
SIZE 1
PAUSE ELAPSED,DELAY
ENDLESSON
APPENDIX E

PAPER-AND-PENCIL EVALUATION
1. Recall the code used to create the BUBBLE sort.
   FOR I := 1 TO N - 1 DO
   FOR J := N DOWNTO I + 1 DO
     IF ARRAY[MAP[J]] < ARRAY[MAP[J - 1]] THEN BEGIN
       TEMP := MAP[J];
       MAP[J] := MAP[J - 1];
       MAP[J - 1] := TEMP
     END
   This code first sends the smallest value to the beginning of the list. Rewrite this code so we send the largest value to the end of the list first.

2. Given the code for BUBBLE sort from problem 1, notice that if the array is already in order the outer loop repeats meaningless. Statements may be added to make this sort stop when the array is in order. Show the code for this modified BUBBLE sort.
3. Recall the code used to create the SELECTION sort.

```plaintext
FOR I := 1 TO N - 1 DO
    BEGIN
        SPOT := I;
        FOR J := I + 1 TO N DO
                SPOT := J;
                TEMP := MAP[ I ];
                MAP[ I ] := MAP[ SPOT ];
                MAP[ SPOT ] := TEMP
        END
```

Rewrite this code so that instead of sending the largest value to the beginning of the list first, we send the largest value to the end of the list last.

4. Rewrite the SELECTION sort given in problem 3 so that the ARRAY will be directly sorted instead of indirectly sorted.
5. Given the following indirect array MAP in ascending order which provides access to a data array,

<table>
<thead>
<tr>
<th>MAP</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

how many elements in the DATA array are physically in between the element which is logically ordered first and the element which is logically ordered last?

6. Given the following indirect array MAP which provides access to a data array, place subscripts into array INV so that

\[ \text{INV}[ \text{MAP}[ \text{subscript} ]] = \text{MAP}[ \text{INV}[ \text{subscript} ]] \].

<table>
<thead>
<tr>
<th>MAP</th>
<th>INV</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
7. A data array was indirectly sorted in ascending order using a selection sort with the resultant map shown below. The same data array was indirectly sorted in ascending order using a bubble sort with the resultant map also shown below.

<table>
<thead>
<tr>
<th>Selection Map</th>
<th>Bubble Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4</td>
<td>1 4</td>
</tr>
<tr>
<td>2 2</td>
<td>2 3</td>
</tr>
<tr>
<td>3 3</td>
<td>3 2</td>
</tr>
<tr>
<td>4 5</td>
<td>4 5</td>
</tr>
<tr>
<td>5 1</td>
<td>5 1</td>
</tr>
</tbody>
</table>

Given the knowledge that sorts are correctly written, provide an explanation of why the two maps are different.
APPENDIX F

PAPER-AND-PENCIL SCORING CRITERIA
The scoring of points for the paper-and-pencil test goes as follows:

| Questions 1-2 | The correct usage of the pair of indirect array references per statement where needed, either $J & J-1$ or $J & J+1$ | 1 point |
| Question 3 | The subscripts remain unchanged | 1 point |
| Question 4 | Modifying the subscripts | 0 points |
| Question 5 | Eliminating the indirect references | 1 point |
| Question 6 | Using only the ARRAY | 1 point |
| Question 7 | Other usage | 0 points |
| Question 5 | Correct answer of 2 elements | 2 points |
| Question 6 | Incorrect answer of 3 elements | 0 points |
| Question 7 | (Subscript 5 - Subscript 2) | 1 point |
| Question 6 | Other answers | 0 points |
| Question 7 | Some subscripts in correct positions | 2 points |
| Question 7 | No subscripts in correct positions | 1 point |
| Question 7 | Sorts treat identical data values differently | 0 points |
| Question 7 | Other answer | 0 points |
Problem Description

You will be presented with a table containing values and subscripts. One element within the table will highlight which is to be accessed.

Below the table will be an array notation in which you must supply the correct subscript so that the highlighted element can be accessed.

For example...

The correct answer is 4.

With this notation...

press RETURN to continue
Problem 1

A value in $W$ is highlighted. What subscript (ROM symbol) provides access to this value?
Problem 2

A value in $V$ is highlighted. What subscript (ROW symbol) provides access to this value?
Problem 3

A value in \( W \) is highlighted. What subscript \((\text{ROM symbol})\) provides access to this value?
Problem 4

<table>
<thead>
<tr>
<th>ROW</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>#</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>G</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

A value in $W$ is highlighted. What subscript (ROW symbol) provides access to this value?
Problem 5

A value in $w$ is highlighted. What subscript (ROM symbol) provides access to this value?

$W[2[\_\_]]$
Problem 6

A value in W is highlighted. What subscript (ROW symbol) provides access to this value?

<table>
<thead>
<tr>
<th>ROW</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

W[ X[  ] ]
Problem 7

A value in \( N \) is highlighted.
What subscript (ROM symbol) provides access to this value?

<table>
<thead>
<tr>
<th>( M )</th>
<th>( X )</th>
<th>( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\( \text{ROM} \)
Problem 8

<table>
<thead>
<tr>
<th>ROM</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

A value in \( M \) is highlighted. What subscript (ROW symbol) provides access to this value?

\[ M(Y[i]) \]
Problem 9

A value in M is highlighted. What subscript (ROW symbol) provides access to this value?

<table>
<thead>
<tr>
<th>ROW</th>
<th>M</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>R</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Problem 10

A value in \( W \) is highlighted. What subscript (RDN symbol) provides access to this value?

<table>
<thead>
<tr>
<th>RDN</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Problem 11

A value in \( |W| \) is highlighted. What subscript (ROM symbol) provides access to this value?

\[
\begin{array}{|c|c|c|c|}
\hline
1 & 3 & 1 & 4 \\
\hline
2 & 5 & 3 & 2 \\
\hline
3 & 2 & 2 & 1 \\
\hline
4 & 4 & 5 & 3 \\
\hline
5 & 1 & 4 & 5 \\
\hline
\end{array}
\]

"W"[2[ ] ];
A value in W is highlighted. What subscript (ROW symbol) provides access to this value?

<table>
<thead>
<tr>
<th>ROW</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
What subscript (ROW symbol) provides access to this value?
Problem 14

What subscript (FDW symbol) provides access to this value?

<table>
<thead>
<tr>
<th>Row</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Problem 15

What subscript (ROM symbol) provides access to this value?
Problem 16

What subscript (ROW symbol) provides access to this value?

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Problem 17

A value in \( W \) is highlighted. What subscript (POM symbol) provides access to this value?
A value in $W$ is highlighted. What subscript (ROW symbol) provides access to this value?
Problem 19

A value in M is highlighted. What subscript (POW symbol) provides access to this value?
Problem 20

A value in $W$ is highlighted. What subscript ($R,W$ symbol) provides access to this value?
Problem 21

A value in \( M \) is highlighted. What subscript (ROW symbol) provides access to this value?

<table>
<thead>
<tr>
<th>ROW</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>X</td>
<td>Z</td>
<td>G</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
<td>H</td>
<td>Z</td>
</tr>
<tr>
<td>X</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Y</td>
<td>A</td>
<td>H</td>
<td>X</td>
</tr>
<tr>
<td>M</td>
<td>D</td>
<td>G</td>
<td>+</td>
</tr>
</tbody>
</table>

\[ M[2[\_\_\_]] \]
Problem 22

A value in $w$ is highlighted. What subscript (ROW symbol) provides access to this value?

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>X</td>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>W</td>
<td>X</td>
<td>B</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>G</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td></td>
<td>H</td>
<td>X</td>
</tr>
<tr>
<td>N</td>
<td>U</td>
<td>G</td>
<td></td>
<td>H</td>
</tr>
</tbody>
</table>
Problem 23

A value in M is highlighted. What subscript (ROW symbol) provides access to this value?
Problem 24

A value in \( M \) is highlighted. What subscript (RDW symbol) provides access to this value?

<table>
<thead>
<tr>
<th>RDW</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>C</td>
<td>W</td>
<td>G</td>
</tr>
<tr>
<td>6</td>
<td>W</td>
<td>Z</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>D</td>
<td>B</td>
<td>S</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>E</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>N</td>
<td>O</td>
<td>M</td>
</tr>
</tbody>
</table>

\[ M[\_\_\_\_\_] \]
Problem 25

A value in \( W \) is highlighted. What subscript (AON symbol) provides access to this value?

<table>
<thead>
<tr>
<th>ROW</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
<td>( _ )</td>
</tr>
<tr>
<td>1</td>
<td>( _ )</td>
<td>H</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>( _ )</td>
<td>Z</td>
</tr>
<tr>
<td>3</td>
<td>( _ )</td>
<td>A</td>
<td>H</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>( _ )</td>
</tr>
</tbody>
</table>

\( W[XY] \)
Problem 26

<table>
<thead>
<tr>
<th>ROW</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

A value in W is highlighted. What subscript (ROW symbol) provides access to this value?
Problem 27

A value in $U$ is highlighted. What subscript (ROW symbol) provides access to this value?
Problem 28

A value in $M$ is highlighted. What subscript (ROW symbol) provides access to this value?

<table>
<thead>
<tr>
<th>$M$</th>
<th>$W$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>E</td>
<td>H</td>
<td>G</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>G</td>
<td>H</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>D</td>
<td>B</td>
<td>H</td>
<td></td>
</tr>
</tbody>
</table>
Problem 29

What subscript (ROW symbol) provides access to this value?
Problem 30

What subscript (ROW symbol) provides access to this value?
Problem 31

What subscript (ROW symbol) provides access to this value?

<table>
<thead>
<tr>
<th>ROW</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>G</td>
<td></td>
<td>H</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>W</td>
<td>O</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>H</td>
<td>Z</td>
</tr>
</tbody>
</table>
Problem 32

<table>
<thead>
<tr>
<th>Row</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>O</td>
<td>G</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>M</td>
<td>B</td>
</tr>
</tbody>
</table>

What subscript (ROM symbol) provides access to this value?
Information for Review of Research Involving Human Subjects
Iowa State University

(Please type and use the attached instructions for completing this form)

1. Title of Project: Effectiveness of Simulation on Indirect Addressing Comprehension

2. I agree to provide the proper surveillance of this project to insure that the rights and welfare of the human subjects are protected. I will report any adverse reactions to the committee. Additions to or changes in research procedures after the project has been approved will be submitted to the committee for review. I agree to request renewal of approval for any project continuing more than one year.

Robert D. Franks 8/1/90
Typed Name of Principal Investigator

Research and Evaluation

Department

Campus Address

Campus Telephone

3. Signatures of other investigators

Date Relationship to Principal Investigator

4. Principal Investigator(s) (check all that apply)
   □ Faculty □ Staff □ Graduate Student □ Undergraduate Student

5. Project (check all that apply)
   □ Research □ Thesis or dissertation □ Class project □ Independent Study (490, 590, Honors project)

6. Number of subjects (complete all that apply)

   □ # Adults, non-students
   □ # ISU student
   □ # minors under 14
   □ # minors 14 - 17
   □ other (explain)

7. Brief description of proposed research involving human subjects: (See Instructions, Item 7. Use an additional page if needed.)

   See Attached

8. Informed Consent: □ Signed informed consent will be obtained. (Attach a copy of your form.)
   □ Modified informed consent will be obtained. (See instructions, item 8.)
   □ Not applicable to this project.
The following are attached (please check):

12. ☐ Letter or written statement to subjects indicating clearly:
   a) purpose of the research
   b) the use of any identifier codes (names, #’s), how they will be used, and when they will be removed (see Item 17)
   c) an estimate of time needed for participation in the research and the place
   d) if applicable, location of the research activity
   e) how you will ensure confidentiality
   f) in a longitudinal study, note when and how you will contact subjects later
   g) participation is voluntary; nonparticipation will not affect evaluations of the subject

13. ☐ Consent form (if applicable)

14. ☐ Letter of approval for research from cooperating organizations or institutions (if applicable)

15. ☐ Data-gathering instruments

<table>
<thead>
<tr>
<th>Anticipated dates for contact with subjects:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>First Contact</td>
<td>Last Contact</td>
</tr>
<tr>
<td>Sept/ 05/1990</td>
<td>Dec/ 20/1990</td>
</tr>
<tr>
<td>Month/Day/Year</td>
<td>Month/Day/Year</td>
</tr>
</tbody>
</table>

17. If applicable: anticipated date that identifiers will be removed from completed survey instruments and/or audio or visual tapes will be erased:
   Dec/ 20/1990
   Month/Day/Year

18. Signature of Departmental Executive Officer Date Department or Administrative Unit
   G. Slatki  Aug. 3  Computer Science
   (Un Slatki was Everyone& s Editor Unit Director)

19. Decision of the University Human Subjects Review Committee:
   ☑ Project Approved  ☐ Project Not Approved  ☐ No Action Required

Patricia M. Keith Date
Name of Committee Chairperson  Signature of Committee Chairperson