The effects of using computer-based models on the learning of computer programming

Warner Keith Smidt
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd
Part of the Engineering Education Commons

Recommended Citation
https://lib.dr.iastate.edu/rtd/8311
INFORMATION TO USERS

While the most advanced technology has been used to photograph and reproduce this manuscript, the quality of the reproduction is heavily dependent upon the quality of the material submitted. For example:

- Manuscript pages may have indistinct print. In such cases, the best available copy has been filmed.
- Manuscripts may not always be complete. In such cases, a note will indicate that it is not possible to obtain missing pages.
- Copyrighted material may have been removed from the manuscript. In such cases, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, and charts) are photographed by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is also filmed as one exposure and is available, for an additional charge, as a standard 35mm slide or as a 17"x 23" black and white photographic print.

Most photographs reproduce acceptably on positive microfilm or microfiche but lack the clarity on xerographic copies made from the microfilm. For an additional charge, 35mm slides of 6"x 9" black and white photographic prints are available for any photographs or illustrations that cannot be reproduced satisfactorily by xerography.
Smidt, Warner Keith

THE EFFECTS OF USING COMPUTER-BASED MODELS ON THE LEARNING OF COMPUTER PROGRAMMING

Iowa State University

University Microfilms International

300 N. Zeel Road, Ann Arbor, MI 48106
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark √.

1. Glossy photographs or pages
2. Colored illustrations, paper or print
3. Photographs with dark background
4. Illustrations are poor copy
5. Pages with black marks, not original copy
6. Print shows through as there is text on both sides of page
7. Indistinct, broken or small print on several pages √
8. Print exceeds margin requirements
9. Tightly bound copy with print lost in spine
10. Computer printout pages with indistinct print
11. Page(s) lacking when material received, and not available from school or author.
12. Page(s) seem to be missing in numbering only as text follows.
13. Two pages numbered. Text follows.
14. Curling and wrinkled pages
15. Dissertation contains pages with print at a slant, filmed as received √
16. Other

____________________________________________________________________

____________________________________________________________________

University Microfilms International
The effects of using computer-based models on the learning of computer programming

by

Warner Keith Smidt

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

Departments: Industrial Education and Technology Professional Studies in Education

Co-majors: Industrial Education and Technology Education (Research and Evaluation)

Approved: Members of the Committee:

Signature was redacted for privacy. 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 Graduate College

Iowa State University
Ames, Iowa
1986
TABLE OF CONTENTS

DEDICATION ix

CHAPTER I. INTRODUCTION 1

  Problem of the Study 4
  Purpose of the Study 4
  Statement of the Need 5
  Assumptions of the Study 10
  Limitations of the Study 11
  Statement of the Research Questions 12

    Research Question I 12
    Statistical Hypothesis I 12
    Research Question II 12
    Statistical Hypothesis II 12
    Research Question III 13
    Statistical Hypothesis III 13
    Research Question IV 13
    Statistical Hypothesis IV 13
    Research Question V 14
    Statistical Hypothesis V 14
    Research Question VI 14

  Definition of Terms 15

CHAPTER II. REVIEW OF THE LITERATURE 18

CHAPTER III. METHODS AND PROCEDURES 28

  Description of the Subjects 28
  Description of the Course 29
  Description of the Computer-Based Lessons 31
  Research Design 41
  Research Procedures 43
  Data Collection Procedures 48
  Description of the Measures Employed 49
### A Priori Statistical Analyses

- Post Hoc Statistical Analyses

### Post Hoc Statistical Analyses

### Custom Computer Programs for Data Analysis

- Closure Program
- Byte Comparison Program
- Fisher's Exact Test Program

### Analyzing Programming Protocols

### CHAPTER IV. FINDINGS

#### Description of Findings - Part One

- Research Question I
- Research Question II
- Research Question III

#### Description of Findings - Part Two

- Research Question IV
- Research Question V

#### Description of Findings - Part Three

- Research Question VI
- Anecdotes

#### Description of Findings - Part Four

- Regression Models
- Problem Solving Approaches

### CHAPTER V. SUMMARY AND DISCUSSION

#### Summary

#### Findings

#### Discussion

- Research Questions I, II, and III
- Research Questions IV and V
- Research Question VI
- Post Hoc Analyses

#### Recommendations for Future Research

- For Both Exploratory and Inferential Research
- For Exploratory Research
- For Inferential Research
Concluding Remarks 127

BIBLIOGRAPHY 129

ACKNOWLEDGEMENTS 133

APPENDIX A. HUMAN SUBJECTS APPROVAL 134

APPENDIX B. LAB SCHEDULE WHILE USING MINIPAS 135

APPENDIX C. COURSE OUTLINE 136

APPENDIX D. SURVEY FORM 137

APPENDIX E. SWAP TEST 142

APPENDIX F. SIMPLE SORT TEST 143

APPENDIX G. COMPARE TEST 144

APPENDIX H. REVERSE TEST 145

APPENDIX I. ARRAY SORT TEST 146

APPENDIX J. EXAMPLE OF A MEMOPS HISTORY FILE 147

APPENDIX K. EXAMPLE OF A MINIPAS HISTORY FILE 149

APPENDIX L. ALGORITHMS AND DATA STRUCTURES EMPLOYED IN CLOSURE PROGRAM 156

APPENDIX M. CLOSURE PROGRAM LISTING 163

APPENDIX N. BYTE COMPARISON PROGRAM 181

APPENDIX O. FISHER'S EXACT TEST PROGRAM 192

APPENDIX P. INCLASS EXAMINATION 196

APPENDIX Q. INTER-ITEM CORRELATIONS AND POINT-BISERIAL CORRELATIONS FOR INCLASS EXAMINATION ITEMS USED FOR CRITERION MEASURE 205

APPENDIX R. UTILITY INCLUDE FILE 207
## LIST OF TABLES

Table 1. Number of Subjects According to Level of Computer Experience and Type of Treatment Received 46

Table 2. Dummy Codes for the Experimental Treatment Variables 52

Table 3. Dummy Codes for the Computer Related Experiences Variables 54

Table 4. Summary of Regression Analysis of Inclass Examinations 62

Table 5. Summary of Inclass Examination Scores by Manipulative Model Treatment Groups 62

Table 6. Summary of Inclass Examination Scores by Memory Treatment Groups 63

Table 7. Summary of Inclass Examination Scores by Manipulative Model and Memory Treatment Groups 64

Table 8. Summary of Obtaining Workable Solutions to Novel Programming Tasks by Manipulative Model Treatment Groups 66

Table 9. Summary of Obtaining Workable Solutions to Novel Programming Tasks by Memory Treatment Groups 67

Table 10. Summary of Regression Analysis of Inclass Examinations Using the Full Model 80

Table 11. Summary of the Six Additional Independent Variables and the Criterion Variable 80

Table 12. Correlation Matrix 81

Table 13. Summary of the Stepwise Forward Regression Results for the Inclass Examination Scores 82

Table 14. Summary of Obtaining Workable Solutions to the Swap Problem 85

Table 15. Summary of the Methods used to Solve the Swap Problem for the Students Who Obtained Workable Solutions 86
Table 16. Summary of the Types of Errors Made by Students Who Did Not Obtain Workable Solutions to the Swap Problem

Table 17. Frequency Counts of the Algorithms Used to Solve the Simple Sort Programming Task by Experimental Group

Table 18. Summary of Obtaining Workable Solutions to the Simple Sort Problem

Table Q-1. Item with Total Score Point-Biserial Correlations for Inclass Examination

Table Q-2. Inter-Item Correlations (Phi Coefficient) for Inclass Examination
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Layout of the Computer Screen for MemOps During Visible Memory Operations</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>Layout of the Computer Screen for MemOps During Invisible Memory Operations</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>List of Exercises Given to Students Using MemOps</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>Layout of the Computer Screen Areas for MiniPas with Visible Memory</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>The Four Main Areas of the Computer Screen in the MiniPas Programming Model</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>Layout of the Computer Screen for Visible Memory MiniPas While a Program is Executing</td>
<td>39</td>
</tr>
<tr>
<td>7</td>
<td>Layout of the Computer Screen for Hidden Memory MiniPas While a Program is Executing</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>The Research Model</td>
<td>42</td>
</tr>
<tr>
<td>9</td>
<td>Color Codes Which Were Assigned to the Four Experimental Treatment Groups</td>
<td>44</td>
</tr>
<tr>
<td>10</td>
<td>Protocol A</td>
<td>69</td>
</tr>
<tr>
<td>11</td>
<td>Protocol B</td>
<td>70</td>
</tr>
<tr>
<td>12</td>
<td>Protocol C</td>
<td>71</td>
</tr>
<tr>
<td>13</td>
<td>Protocol D</td>
<td>72</td>
</tr>
<tr>
<td>14</td>
<td>Protocol E</td>
<td>72</td>
</tr>
<tr>
<td>15</td>
<td>Protocol F</td>
<td>73</td>
</tr>
<tr>
<td>16</td>
<td>Protocol G</td>
<td>73</td>
</tr>
<tr>
<td>17</td>
<td>Protocol H</td>
<td>73</td>
</tr>
<tr>
<td>18</td>
<td>Protocol I</td>
<td>74</td>
</tr>
<tr>
<td>19</td>
<td>Protocol J</td>
<td>74</td>
</tr>
<tr>
<td>20</td>
<td>Protocol K</td>
<td>74</td>
</tr>
</tbody>
</table>
DEDICATION

To my wife Pam and my daughters
Sarah Jean, Erin Grace, and Abby Johanna

in appreciation for their many sacrifices during this endeavor
CHAPTER I. INTRODUCTION

One of the difficulties in teaching novices how to program a computer is to describe -- at the appropriate level of detail -- the properties of the computer. The purpose of a computer program is to control the computer in order to accomplish a given task. By writing and successfully implementing a computer program, the student is indirectly performing the task via the program "telling" the computer what to do. Therefore, before the novice can program the computer, he or she must understand what the machine can do and how it manages to do it.

The process of developing a working or functional knowledge of programming hinges on (1) the properties of the programming language, (2) the characteristics of the learner, and (3) the instructional techniques. All three of these are interdependent. This study dealt with the latter two. This does not imply, however, that the properties of the programming languages were not important. On the contrary, the structure of the language must support good programming practices (Sime, Green, and Guest, 1973, 1977; DuBoulay and O'Shea, 1981; Jensen and Wirth, 1974; Coombs and Alty, 1981).

Numerous researchers have studied the programming process and the results have been mixed (Sheil, 1981; Schneiderman and Mayer, 1979; Hooper, 1986b). "It is almost universally agreed that empirical work on the nature and acquisition of programming skills has been less than successful" (Coombs, Gibson, and Alty, 1981, p. 145). The programming tasks which have been studied include planning, composition, coding,
comprehension, debugging, modification, and the learning of new programming skills.

The majority of the research on programming skills was motivated by the need to improve the efficiency of large-scale software projects involving teams of programmers. The main concern was not how students learned to program per se but rather the economics of programming from a production and business perspective (Coombs, Gibson, and Alty, 1981; Shneiderman, 1980). Questions like: Why do learning differences between programming students exist? Was the learning meaningful? Did the learning facilitate future learning? went unanswered (Schneiderman and Mayer, 1979; Pea and Kurland, 1984; Sternberg, 1985a).

Not only is little known about the cognitive aspects of how novices learn to program; likewise, little is known about the cognitive impact of computer-based lessons on student learning. A considerable amount of research has compared the effect of "conventional" instruction to that of computer-based instruction. This type of research naively assumed that the medium rather than some specific attribute or quality of the medium, affects learning. When all other factors are held constant -- save the medium itself -- essentially no learning effects have been observed. Only specific, relatively unique features of the medium make a difference in learning outcomes (Salomon and Gardner, 1986). Some of the unique features of computer-based learning which demand further investigation are the interaction between user and computer, learner control and self pacing, the adaptation of computer-based lessons to various student learning styles, the student's ability to generate and test ideas, immediate feedback, idea or concept
modeling, and the ability to simulate real world systems (Clark, 1985).

This research study examined the effects of using computer-based lessons on the learning of a technologically complex system. In particular, this research study examined the effect on novice learning of computer programming when the student can manipulate a dynamic, computer-based model of the technological system which was to be learned. Empirical studies by Mayer (1975, 1981) indicate that three conditions must be met before meaningful learning will occur: (1) the material to be learned must be received by the learner, (2) the learner must have prerequisite knowledge on which to anchor the new learning (called the assimilative set), and (3) the learner must actively connect the prerequisite knowledge to the new material being learned (called activation of the assimilative set).

Based on Mayer's assimilative theory, Thomas, Boysen, and Hooper (1985) developed two computer-based lessons which serve as concrete models for student learning. These lessons were designed so that students could manipulate and test their ideas by interacting with the dynamic computer-based model. The concept which was chosen to be the model topic was data storage in computer memory. This concept was chosen because beginning students must have a meaningful understanding of data storage in computer memory before they can be successful computer programmers.

The names of these two lessons are MemOps (short for Memory Operations) and MiniPas. The purpose of MemOps was threefold: (1) to serve as a dynamic model of computer memory which was manipulated by the learner, (2) to facilitate algorithm development, and (3) to lay a
foundation for the use of arrays. MiniPas was designed to provide a learning environment for the beginning programmer which was more transparent than the traditional programming environments. The use of these two lessons was an integral part of this research study.

Problem of the Study

The problem of this study was to explore what effects the use of manipulative, dynamic computer-based models had on the meaningful learning of computer programming by novices.

Purpose of the Study

The purpose of this study was fourfold:

1. To help educators better understand student learning when manipulative, dynamic computer-based models were used in the instructional sequence.
2. To identify programming practices which beginning programming students used which may suggest programming practices not typically used by experienced or expert programmers.
3. To identify features of computer-based learning systems which affected student learning.
4. To provide developers of computer-based instructional materials with information which would be useful in the design of computer-based lessons.
Statement of the Need

In a paper presented at the 1966 Symbol Manipulation, Language, and Techniques Conference, Monrad-Krohn stated that the properties of human beings could no longer be forgotten by designers of computer languages. Prior to 1966, human factors engineering did not influence the design of computer software due to technical limitations.

Early research on programming focused on the features of the language and was motivated by the need to improve the efficiency of large scale software projects involving teams of programmers (Weinberg, 1971). Only recently (late 1970s) has more emphasis been directed toward studying programming skills themselves (Coombs and Alty, 1981).

The shift in the type of user has also forced the need to understand how novices program. People who have had no computing backgrounds, but are expert in other areas such as medicine and social science, are needing to use the computer in their jobs (Coombs et al., 1981; Shneiderman and Mayer, 1979). Coombs et al. (1981) state that

It is almost universally agreed that empirical work on the nature and acquisition of programming skills has been less than successful. This is largely because the studies often lack adequately defined variables for manipulation and measurement. A further problem is the lack of adequate experimental designs to cope with the complexity of programming, which does not appear to readily decompose into skills which can be studied separately. It is therefore proposed that a new approach is necessary. We would argue that considerably more attention should be paid to developing good descriptions of programming behavior (p. 145).

In designing an interactive, computer-based tutorial system to teach BASIC, Barr, Beard, and Atkinson (1976) implemented tracing
facilities "to assist the student in conceptualizing the execution of his program." The tutorial program would trace the flow of the program execution and would trace the values of up to six variables during run-time. The tracing features could be turned on or off at the discretion of the student. The rational for incorporating these features was because the researchers noticed that beginning programmers had difficulty understanding the flow of execution and the dynamics of memory. Barr, Beard, and Atkinson (1976) never investigated or reported the effects on student learning of using the trace of the flow of program execution and the trace of the six variables.

Other computer-based systems implemented to teach programming which included tracing features are described by Shapiro and Witmer (1974), Nievergelt et al. (1978), and Schweppe (1973). Although these systems were for other languages, in all cases, the researchers felt the tracing features would be useful learning tools for the programming student, but no data were collected to ascertain whether these features were useful or not.

Another study which mentioned the use of tracing features dealt with the debugging of programs by professionals. The goal of the study was to determine what types of errors were the most difficult to find. The subjects were given program listings and were told to find the semantic or logic error. The various tools which the programmers had at their disposal included example input data, output results of the program with the error, correct output results, and interactive debugging facilities. Two of the debugging features were flow of control tracing and variable value tracing. In reporting the results,
Gould (1975) noted that the debugging facilities were seldom used (used on 15% of the listings); the input data was rarely used; most subjects used the output data, but the use of the output data did not improve debugging performance. Gould only reported how often the debugging facilities were used, he did not examine how or when the programmers used them.

Little empirical research into the cognitive factors involved in programming has been done (Seidel, Anderson, and Hunter, 1982; Mayer, 1979; Ledgard, 1979, as cited in Soloway et al., 1982; Mayer, 1975; Dalbey and Lynn, 1985). In an introductory programming course, Soloway et al. (1982) collected on-line protocols as students interacted with the Pascal system to solve homework problems. Results were unsettling. Students did not seem to understand (or trust) the iteration constructs. Some students assumed the construct performed more actions than actually are performed (e.g., the While construct incrementing the index variable). "We speculate that, in fact, students do not have mental models of the primitives which compose these higher level constructs" (Soloway, 1985, p. 183).

The speculation by Soloway et al. (1982) paralleled that of Mayer (1979) who suggested there are several levels of programming knowledge which must be explicitly taught -- not just the two levels of (1) statement definition and grammar, and (2) programs which incorporate the just learned statements. Statements must be broken down into transactions; transactions being the lowest level of programming knowledge required. And how statements are "chunked" together to do certain tasks must be taught. Mayer's and Soloway's research tended to
support the need to explicitly teach the various levels of programming knowledge when teaching novices. Both researchers stressed the need for additional research to verify and expand upon what they have done (Mayer, 1979, 1982; Soloway et al., 1982).

In summarizing the presentations on "Cognitive Research and Solving Problems Using the Computer" made by Mayer, Soloway, Lochhead, and John Clement at the conference entitled "National Goals for Computer Literacy in 1985" (held in 1980), the following observations were given by several researchers who were in attendance.

Olson (1982) noted that

They [the cognitive theories underlying the cognitive research] have been based primarily on informal observation, anecdotes, opinions, biases and briefs. Almost no research has been discussed. . . . For without knowledge of the cognitive system to be made literate, it is going to be difficult to know how to proceed (p. 188-189).

. . . first of all, there is not enough theory. What kinds of psychological mechanisms are being examined? What are the skills, the relevant knowledge? How do the processes work? Under what condition does transfer occur and when doesn't it? . . . Without clear theory, we cannot advance (p. 191).

Soloway et al. (1982) stated "that mental models of the primitives which compose these higher level constructs are needed" (p. 181).

Seidel, Anderson, and Hunter (1982) stated:

[In reference to the cognitive research] One point not discussed sufficiently is how we evaluate. Often our evaluations results show nothing much has happened. The fact of the situation is a lot has happened but we don't know much about how to evaluate or measure it (p. 199).

Other observations made by researchers investigating the cognitive aspects of programming (separate from the above mentioned conference)
which re-enforced the need for this research are given below.

Shneiderman and Mayer (1979) claim:

The intuitions and experience of expert programmers and programming language designers are no longer appropriate for developing facilities to be used by novices with varied backgrounds (p. 220).

Thomas Dwyer (1974), in discussing various heuristic teaching strategies to enrich educations, stated:

A problem that sometimes confronts an educational system based on student-controlled technology is the request to "prove" its worth through evaluation based on standardized tests. Using such evaluation can be a misleading exercise, since standardized tests (and the test-makers who devise them) favor the outcomes of dual mode instruction; both questions and answers are influenced by the models of the test-makers. Papert suggests that using such test to evaluate an innovative instructional system concerned with eliciting new models, is like evaluating the potential of precision electric motor by measuring its ability to drive a cart meant to be drawn by horses (p. 151).

A considerable amount of research has been done which has examined the type of errors made by programmers, the novice programmer versus experienced programmer, the type of errors with respect to the language used, the effect of program size and the types of errors made, the use of debugging aids (e.g., sample input and output, on-line debuggers). The majority of this empirical research did not examine the cognitive aspects of the programmer.

The need for this study can be summarized in a statement made by Robert Sternberg (1985b) while discussing the key phenomena of human intelligence: "As important as it is to know what individuals are doing, it is even more important to know why they are doing it" (p. 127).
Assumptions of the Study

This study was based upon the following assumptions:

1. It was assumed that the number of errors and the test scores were normally distributed, random, and independent. Homogeneity of variance was also assumed. In other words, all sample populations were assumed to be from the same population.

2. It was assumed that no interaction (social, academic, or otherwise) occurred among students outside of the experimental setting which affected the results of the study.

3. It was assumed that the lab presentations given on consecutive days to the two different lab sections did not differ in any manner affecting the experiment.

4. Since each lab section was physically separated into control group (hidden memory) and treatment group (visible memory) during all lab sessions, it was assumed that the lab presentations given by the two lab instructors did not differ in any manner affecting the experiment.

5. It was assumed that random assignment of the two lab instructors to the control group (hidden memory) and treatment group (visible memory) controlled for the differences between the individual lab instructors.

6. Two different lab sites were used in order to separate the control group (hidden memory) and treatment group (visible memory). The two lab facilities (physical layout, terminals, printers, method
of connecting terminals to the host minicomputer) were not identical, but similar. It was assumed that rotating the two groups between these two labs controlled for the lab differences.

Limitations of the Study

The limitations of this study were as follows:

1. The sample was limited to Industrial Education and Technology students at Iowa State University.
2. Since the study occurred during actual course instruction, no control over questions asked of students, responses given to questions, or time devoted to raised questions could be strictly controlled.
3. The response time of the shared VAX 11/780 system varied with job loads imposed by other users around the university and was beyond the control of the investigator.
4. Random selection of subjects for treatment group membership did not guarantee group equality. Matching on variables related to the dependent variables was not possible. Group differences which existed due to extraneous variables could have had an effect on the experimental results.
5. Type II control was not specified due to the restrictions of working with available classes of fixed size.
Statement of the Research Questions

Research Question I

Will there be a difference in learning between students using the manipulative model and students who do not use the manipulative model?

Statistical Hypothesis I

\[ H_0: \mu_{\text{MemOps}} = \mu_{\text{Placebo}} \]
\[ H_A: \mu_{\text{MemOps}} \neq \mu_{\text{Placebo}} \]

The criterion variable for this hypothesis was the inclass examination scores.

Research Question II

Will there be a difference in learning between students using visible memory and students who use the invisible memory?

Statistical Hypothesis II

\[ H_0: \mu_{\text{Visible}} = \mu_{\text{Hidden}} \]
\[ H_A: \mu_{\text{Visible}} \neq \mu_{\text{Hidden}} \]

The criterion variable for this hypothesis was the inclass examination scores.
Research Question III

Will there be an interaction effect between student membership in the manipulative model groups (MemOps or placebo groups) and student membership in the memory groups (visible or hidden groups)?

Statistical Hypothesis III

\[ H_0: u_{\text{row}} + u_{\text{col}} + u_{\text{row}\times\text{col}} - u_{\text{total}} = 0 \]
\[ H_A: u_{\text{row}} + u_{\text{col}} + u_{\text{row}\times\text{col}} - u_{\text{total}} \neq 0 \]

The criterion variable for this hypothesis was the inclass examination scores. Row refers to MemOps or Placebo groups. Col refers to Visible or Hidden Memory groups. Row\times Col refers to the interaction effect for a given cell at a row or column.

Research Question IV

Will there be a difference between the proportion of MemOps students who obtain workable solutions to novel programming tasks and the proportion of Placebo students who obtain workable solutions?

Statistical Hypothesis IV

\[ H_0: p_{\text{MemOps}} = p_{\text{Placebo}} \]
\[ H_A: p_{\text{MemOps}} \neq p_{\text{Placebo}} \]

The criterion variables for this hypothesis were dichotomous scores which indicated whether or not students obtained workable
solutions to various novel programming tasks.

**Research Question V**

Will there be a difference between the proportion of visible memory students who obtain workable solutions to novel programming tasks and the proportion of hidden memory students who obtain workable solutions?

**Statistical Hypothesis V**

\[ H_0: \, p_{\text{Visible}} = p_{\text{Hidden}} \]
\[ H_A: \, p_{\text{Visible}} \neq p_{\text{Hidden}} \]

The criterion variables for this hypothesis were dichotomous measures which indicated whether or not students obtained workable solutions to various novel programming tasks.

**Research Question VI**

What types of programming practices do beginning programmers use that may suggest programming practices not typically used by experienced or expert programmers?

This research question is descriptive in nature rather than inferential; thus, there is no statistical hypothesis for this research question.
Definition of Terms

The terms listed below were used in this study and require definition.

1. **Courseware Authoring System (CAS)** is the instructional lesson delivery system developed by the Digital Equipment Corporation. CAS runs on the VAX minicomputer.

2. **Clerical errors** are student errors which are due to carelessness or accident in the coding process, such as misspelling a variable or keyword (DuBoulay and O'Shea, 1981).

3. **Digital Authoring Language (DAL)** is the high level authoring language which was used to write the two computer-based lessons (MemOps and MiniPas) on VAX.

4. **Hidden Group** (versus the Visible Group) consisted of the students who were not able to see the contents of memory when they used the MiniPas program.

5. "**Logical errors** concern the mapping from the problem to the program" (DuBoulay et al., 1981, p. 237). A program which contains a logic error compiles without any syntax errors and the program runs successfully without encountering any semantic errors, but the output results of the program are not correct for the problem at hand.

6. The students in the **MemOps Group** (versus the Placebo Group) used the manipulative computer model, MemOps, prior to formal computer language instruction. The MemOps group was one of the
experimental treatment groups.

7. **Memory Operations** (MemOps) is a computer-based lesson designed to teach students the concepts of memory operations by requiring the learner to actively manipulate a dynamic model of computer memory. The use of MemOps by half of the students versus the non-use of MemOps by the remaining students was one of the independent variables in this study.

8. **MiniPas** is a computer-based environment for learning Pascal programming. The visibility of the memory area in MiniPas was one of the experimental main effects. Students in the visible treatment group saw the values of variable in the memory area, but students in the hidden group did not see the memory area.

9. The **Placebo Group** (versus MemOps Group) included the students who used a substitute or placebo computer-based lesson instead of using MemOps. The Placebo Group was the control group for one of experimental treatments.

10. **Programming protocols** refer to the original programs which were written by the students as they were solving a programming problem.

11. "**Semantic errors** try to make the computer carry out impossible or contradictory actions, though the expression is syntactically correct" (DuBoulay and O'Shea, 1981, p. 151). For example, reading or writing to a file which is not open is a semantic error.

12. **Stylistic errors** "make a program hard to comprehend or very inefficient, though it runs correctly" (DuBoulay and O'Shea,
13. "Syntax errors are incorrect expressions in the language that cause the compiler or interpreter to generate an error message" (DuBoulay and O'Shea, 1981, p. 151). Misuse of delimiters, failure to declare variables, and data type mismatches are examples syntax errors.

14. VAX is a minicomputer manufactured by Digital Equipment Corporation. The two computer-based lessons, MemOps and MiniPas used in this study were implemented on the VAX computer at Iowa State University, Ames, Iowa.

15. The Visible Group (versus the Hidden Group) refers to the students who saw the contents of memory at all times while using MiniPas.
Prior to the advent of the microcomputer in the middle 1970s, the use of computers was essentially restricted to computer scientists, engineers, census takers, and other professionals who had large data processing requirements. The low cost of the microcomputer has made computing more widespread. In fact, computing is now beginning to be adopted by schools and universities as a basic subject. Business and industry are seeking employees who can apply the use of computers in the everyday workplace. This rise in the need for students and employees with computing skills has placed a demand for computing instruction at a time when little is known about the nature of computing skills and little is known about the effective methods for teaching these skills (Coombs and Alty, 1981).

Industrial Education and Technology (IETD) students are technologically knowledgeable in various areas (e.g., materials and processes, graphic communications, power and energy) associated with the manufacturing and construction technologies. IETD students will need to interact with computers both in college and in their future jobs; but they are not planning to be professional programmers. An important issue, therefore, concerns how to foster meaningful learning of computer concepts for these students.

Schneiderman and Mayer (1979) purport that there are at least two types of knowledge required for programming: semantic and syntactic. Semantic knowledge consists of general programming concepts, is independent of specific programming languages, is acquired primarily
through meaningful learning, and is essential for problem solving.
Syntactic knowledge, compared to semantic knowledge, is more precise,
detailed, and arbitrary (thus more easily forgotten), is language
dependent, is acquired largely through rote learning, and is used during
coding and implementation.

Mayer (1979, 1981) and DuBoulay and O'Shea (1981) claim that
novices must know how computers work before meaningful learning of
programming can occur. DuBoulay and O'Shea use the terms "black box"
and "glass box" to describe the two approaches of learning how to
program. The black box approach hides the internal workings of the
computer from the user. From the student's perspective, the computer
performs its task as if by magic. The glass box approach provides a
pictorial or written commentary which allows the novice to look inside
the black box and "see" what is happening. The glass box approach
reveals to the learner the notional machine. "The notional machine is
an idealized, conceptual computer whose properties are implied by the
constructs in the programming language employed. That is, the
properties of the notional machine are language, rather than hardware,
dependent" (DuBoulay et al., 1981, p. 237). Mayer (1979, 1981) and
DuBoulay and O'Shea (1981) contend that the glass box approach would
facilitate meaningful learning; whereas, the black box approach
essentially forces the learning to be arbitrary and rote.

Mayer's (1981) research has shown that the meaningfulness of
student learning of computer programming is enhanced significantly when
the students receive a static, concrete model of a computer (e.g.,
input/output windows, a memory scoreboard) before formal language
instruction is given (as opposed to students who received the concrete model after formal language instruction). The students who received the concrete model before formal language instruction had a higher level of semantic knowledge as revealed by the superior performance on novel programming tasks given subsequent to instruction. The students, however, who received the concrete model after formal language instruction had higher levels of syntactic knowledge.

"Meaningful learning is the process in which the learner connects new material with knowledge that already exists in memory" (Mayer, 1981, p. 121). Three conditions must be met in order for meaningful learning to occur. Learners must (1) receive or pay attention to the new information, (2) possess appropriate old knowledge or prerequisite concepts to which the new material can be anchored, and (3) actively assimilate or connect the new knowledge to the old.

Ausubel (1963) distinguished between two types of learning: meaningful and rote. "Meaningful learning takes place if the learning task can be related in nonarbitrary, substantive fashion to what the learner already knows, and if the learner adopts a corresponding learning set to do so" (Ausubel, 1963, p. 18). Meaningful learning is an active cognitive process by the learner. If there are discrepancies or conflicts between the new material and existing knowledge, the learner must resolve these differences. Therefore, the new material may have to be transformed to fit existing mental structures; or the existing mental structures may have to be reorganized to reconcile the discrepancies.

If the learner's intention is to memorize verbatim, then learning
will be rote; in other words, the learner does not employ a meaningful
learning set. Likewise, the learning will be rote if the material to be
learned is purely arbitrary; thus, the material cannot be integrated
into the existing cognitive structure (Ausubel, 1960, 1963).

Similar distinctions between meaningful and rote learning have
been made by other researchers. Mayer (1981), who founded his thinking
on Ausubel's work, defines the term understanding as

the ability to use learned information in problem solving
task that are different from what was explicitly taught.
Thus understanding is manifested in the user's ability to
transfer learning to new situations (p. 122-123).

Gestalt psychologists made the same distinction between two ways of
learning how to solve problems -- rote (or senseless) learning versus
understanding (or structural learning).

A good classroom teacher intuitively knows there is a difference
between meaningful and rote learning, but may not be able to articulate
the difference. Consider the following everyday classroom situation. A
student has just completed a problem solving task. The teacher asks the
student, "Your answer is correct. Now, can you tell me why your answer
is correct?" This teacher realizes that the correct student response
does not necessarily reflect meaningful learning on the student's
behalf. The purpose of the teacher's question is to determine if the
student was rote ly solving the problem or did the student truly
understand the problem (or was it just a lucky guess).

Brownell and Moser found that students who were taught subtraction
while manipulating bundles of sticks performed better on transfer tasks
(i.e., more complicated subtraction problems) than did students who
rotely learned the rules of subtraction. In interviews with the children, it was found that the students either explained the process of subtraction in terms of the sticks they had manipulated or they had invented mental models to help them understand the subtraction procedures (Mayer, 1981).

A parallel can be drawn between the concept of meaningful versus rote learning and Robert Sternberg's (1985a, 1985b) triarchic theory of human intelligence. The triarchic theory consists of three subtheories:

1. componential subtheory deals with the internal, mental world of the individual; consists of metacomponents, performance components, and knowledge acquisition components;
2. contextual subtheory deals with the external, environmental world of the individual; consists of adaptation, selection, and shaping;
3. experiential subtheory deals with the individual experiences as they relate to the internal and external worlds; consists of the ability to deal with novelty and the ability to automatize processing.

The three parallels are (1) the prerequisite knowledge is the individual's internal, mental world; (2) the new information to be learned is the external, environmental world of the individual; and (3) meaningful learning is dependent on the experiences which the individual has to relate the internal and external worlds together.

Central to the concept of meaningful learning is the activation of prior learning in order to assimilate or anchor the new material to be
learned. If the learner does not possess the necessary prerequisite knowledge or if the prerequisite knowledge is present but not activated, the resulting learning will be rote -- a mere mental cataloging of the material. Therefore, in order to make the learning of computer programming meaningful to students, the students will have to activate the prerequisite knowledge.

Since meaningful learning is an active process, collecting benchmark measurements of students by using pretest and posttest experimental techniques is inadequate. Several researchers have collected programming protocols to determine the types of errors which students make while programming (Soloway et al., 1982). Programming protocols refer to the original programs which were written by the students as they were solving a programming problem. Hooper (1986b) collected protocols and analyzed them to determine what approaches and algorithms novices employed to solve programming problems.

The subjects in the Hooper study were students in a beginning IEDT programming course. Half of the students were randomly assigned to a manipulative model group and the other half were assigned to a placebo group. The students in the model group were told to do computer-based lesson (called MemOps) which was designed to provide the novice programmers with discovery learning experiences dealing with the movement of data in computer memory. The students in the placebo group did a computer-based problem solving activity which had nothing to do with computer programming concepts. Both of the groups did these problem-solving exercises prior to any formal computer programming language instruction. After the MemOps and placebo experimental
treatments, the "normal" classroom instruction proceeded. The following Pascal language topics were covered (for both the control and treatment groups) after the MemOps experience: keyboard input, screen output, variables and assignment statements, and the binary branching control structure.

After completing the instructional unit on the binary branching control structure, all students were given two novel programming tasks which they were to solve. Students wrote their initial solutions on paper and then completed (and/or debugged) their original solutions by using MiniPas, an interactive programming environment for beginning Pascal programmers. The original solutions on paper and the programming protocols as collected by MiniPas were later analyzed. Several weeks later, just after the concept of arrays was covered, the students again were required to solve a novel, programming task incorporating arrays. At this time, they also completed a paper and pencil test. Again, the programming protocols were collected for analysis.

A summary of Hooper's findings which are relevant to this study follows.

1. The algorithms which the students choose to use when attempting to solve various programming tasks was significantly influenced by the use and non-use of MemOps.
2. Novices did a poor job of relating recently learned concepts to new programming applications.
3. The intuitive mental models of computer memory which novice programmers possessed were not accurate.
4. When working on programming problems, students made relatively few changes in the algorithm or overall approach to the problem but made considerably more changes in syntax and logic.

5. Novice programmers had difficulty separating the way humans process information and the way computers process information.

6. The use of MemOps did not facilitate the students' ability to find syntax errors, to hand-execute programs, nor to obtain workable solutions to given programming tasks.

Mayer's (1975) research indicated that students who used a concrete model prior to formal instruction performed better on novel programming tasks than the students who did not have the concrete model prior to formal instruction. Whereas, the latter group of students was better at syntax than was the former group. Hooper's findings did not refute Mayer's findings. MemOps did have an effect on the type of algorithm which students choose to use when solving a programming task and MemOps did not affect the students' ability to find syntax errors. In other words, the use of the manipulative model, MemOps, had an effect on a high level skill, algorithm selection, but had no effect on a rote skill, finding syntax errors.

Pea (1986) used a different approach than Hooper to determine what type of misconceptions novice programmers possessed. Students, who had one semester of programming, were presented with segments of various programs. The students were then asked to explain what each segment of code would do. Three classes of conceptual misunderstandings or "bugs" were identified: parallelism, intentionality, and egocentrism. The
parallelism bug occurred where conditional statements which were outside of (independent of) loops. In other words, the students thought that the boolean conditions in IF THEN statements remained active for the duration of the program's execution. The intentionality error was the result of the student looking ahead and thinking an event occurring before it actually was suppose to occur. The third type of error, egocentrism errors, was when the student went beyond what was actually occurring in the segment of code and essentially created what was to occur.

On the basis of this review of the literature, it is the conclusion of this writer that little is known about the nature of how students learn computing skills and little is known about effective methods for teaching these skills. Likewise, little is known about the cognitive processes which are occurring when a student dynamically interacts with a computer-based lesson. And, even less is known about how to quantify and document what is occurring cognitively when a student is learning to program or when a student is dynamically interacting with a computer-based lesson.

Research is required which will first identify what are the cognitive processes which occur when a novice learns to program and when a student interacts dynamically with a computer-based lesson. After the cognitive processes are identified, valid and reliable techniques must be developed to quantify the identified cognitive processes. Finally, after the cognitive processes are identified and after the techniques have been developed to quantify the cognitive processes which are taking
place, then the important issue of determining why the cognitive processes are occurring can be addressed.
Description of the Subjects

The population for this study consisted of industrial education and technology students enrolled in a first semester computer programming course. The sample, however, was confined to those students enrolled in the Computer Applications in Industrial Education and Technology course (IEDT 216) at Iowa State University, Ames, Iowa. There were two sections of this class during the spring semester of 1986. This group of subjects was chosen simply as a convenience sample.

A total of 45 subjects were initially enrolled in the course. Two students dropped the course after the first two weeks of the course; therefore, the initial sample size was 43. A third student stopped coming to class after attending the first three weeks of the semester. A fourth and fifth students' class attendances were so poor (less than 50%) that the data collected for these two students were incomplete. Of the remaining 40 students, two students did not attend class during the final examination periods; therefore, the data for these two students is complete, except for the final examination data. Complete data were collected for a total of 38 subjects.

The following demographic description of the study participants is based on the original 43 students. All but one of the subjects were males. Thirty-nine (91%) of the students were IEDT majors. Of the 39 IEDT majors, 23 (59%) were taking the course to meet a computer
programming requirement, two students (5%) took the course because it was a prerequisite for future course work, and the remaining 14 students (36%) took the course as an elective. It was an elective course for the four non-IEDT majors. The average age of the subjects was 22.9 years with the youngest student being 20 years of age and three students were over the age of 28 (33, 35, and 36).

Description of the Course

The course description as given in the Iowa State University General Catalog (1985) is as follows:

Provides a working knowledge of microcomputers and their application in industry and teaching industrial arts. Applications to energy and power, graphic communications, and materials and processes (p. 162).

IEDT 216 is a three semester hour course which meets three times a week: twice a week for a two hour lab session and once a week for a common, one hour lecture. IEDT 216 or a computer science Pascal programming course is a requirement for all IEDT industrial option students. It is not a required course for IEDT teaching option students; however, the IEDT department strongly recommended that teaching option students take the course as an elective (Sherick, 1986).

Two sections of IEDT 216 were offered during the spring of 1986. Labs for Section A met from 10 to 12 A.M. on Mondays and Wednesdays. Labs for Section B met from 8 to 10 A.M. on Tuesdays and Thursdays. Both sections had a common, one hour lecture at 10 A.M. on Fridays.

In the past, the first topic covered in IEDT 216 was the use of a
wordprocessor. The remainder of the course was dedicated to learning how to program in Pascal with emphasis in IEDT programming applications. A homework room was provided so students had access to a microcomputer outside of the scheduled lab time to complete assignments. Many students had access to microcomputers at their campus residences and therefore did not have to compete with other College of Education students using the the same microcomputer homework room.

The primary purpose for preceding the Pascal instruction with wordprocessing was to provide a tie from past student experiences (writing and typing papers) to the new learning (computer programming) which was forthcoming. In the process of learning how to wordprocess, many concepts inherent to programming could be addressed; e.g., loading a program into memory, saving a data file from memory to disk and later loading it back into memory again, and editing a text file in primary memory.

Since the wordprocessing experience would help students learn many of the same concepts (i.e., memory operations) which were parallel to the same concepts which MemOps and MiniPas were to affect, the word processing activity was delayed until the second half of the semester. Not only was the potential confounding effect of wordprocessing removed from the experiment, but students who had microcomputers at their campus residences would not have an unfair advantage over students who did not have such access to a microcomputer outside of class hours.

To reduce the possible effect on student learning that the availability of computing facilities may have, the Pascal language instruction for the first half of the semester was done in the MiniPas
programming environment. Since a special terminal (a DEC GIGI or a
VT241 connected to the VAX minicomputer) was required to run MiniPas,
all students in the class had equal access to the computing facilities
outside of class.

Description of the Computer-Based Lessons

The problem of this research study was to determine what effect the
use of two computer-based lessons had on the meaningful learning of
computer programming by novices. The names of these two lessons are
MemOps (short for Memory Operations) and MiniPas. Since the use of
these two lessons was an integral part of this research study, an
explanation of each lesson is required.

The purposes of MemOps are (1) to serve as a dynamic model of
computer memory which can be manipulated by the learner, (2) to
facilitate algorithm development and (3) to lay a foundation for the use
of arrays. MemOps provides the student with six memory locations (X[1],
X[2], X[3], X[4], X[5], and Z) which contained randomly assigned integer
values. The layout of the computer screen for MemOps is shown in Figure
1. The student is prompted with a series of exercises which forces the
learner to interact with and manipulate the contents of the memory
locations in order to successfully complete the exercises.

The series of exercises is divided into two parts: operations on
visible memory and operations on invisible memory. During visible
memory operations, the student can see on the computer screen at all
times what the values are which are stored in each memory location. The
computer screen, as it appears for the visible memory operations, is

INSTRUCTION CODE
> move x[21] to z

MEMORY

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>???</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>???</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>???</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>???</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Type check to evaluate your solution.

Figure 1. Layout of the Computer Screen for MemOps During Visible Memory Operations

shown in Figure 1. During the invisible memory operations, values are still stored in the memory locations, but they are covered up so that the student can not see what the contents of the six memory locations are. Figure 2 shows how the computer screen appears during the invisible memory operations.

The student is required to perform nine exercises: four on visible memory and five on invisible memory. The specific exercises are listed in Figure 3. The nine exercises are listed in the order in which the student performs the exercises. When the student is ready to do an
**EXERCISE:** Sort the array $X$ into ascending order. The smallest value should be in $X[1]$. The largest value should be in $X[5]$.

**INSTRUCTION CODE**


The value in $x[1]$ is less than the value in $x[2]$.

**MEMORY**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td></td>
</tr>
<tr>
<td>$X$</td>
<td>$Z$</td>
</tr>
</tbody>
</table>

Type check to evaluate your solution.

Press $\textlangle$ help $\textrangle$ restart $\textlangle$ menu

---

Figure 2. Layout of the Computer Screen for MemOps During Invisible Memory Operations

Exercise, MemOps fills the appropriate memory locations with random values and then presents the student with the exercise to be performed. To complete each exercise, the student must give special commands to MemOps.

For the visible memory exercises, the student has three commands which can be issued in order to accomplish the task: move, check, restart. For example, if the student enters "MOVE $X[1]$ TO $Z$", MemOps draws an arrow from $X[1]$ to $Z$ and the current value in $X[1]$ is copied.
### Operations on Visible Memory

- Move the smallest value in X to Z
- Move the largest value in X to Z
- Swap the contents of X[1] with X[2]
- Sort X into ascending order
- Sort X into descending order

### Operations on Invisible Memory

- Move the smallest value in X to Z
- Move the largest value in X to Z
- Sort X into ascending order
- Sort X into descending order

---

**Figure 3. List of Exercises Given to Students Using MemOps**

into Z. After giving as many "move" commands as the student deems necessary to complete the exercise, the student types in the "check" command. MemOps then tells the student whether or not the task was successfully accomplished. If the task was not completed successfully, the student repeats it over again with a new set of random numbers placed in the memory locations. The "restart" command allows the student to start the current exercise over again at the beginning with the same set of values in the memory locations. The student can give the "restart" command at any time prior to giving the "check" command for a particular exercise.

When the student does the invisible exercises, the "compare" command is also available. For example, the command "COMPARE X[3] WITH X[5]" would trigger MemOps to respond with a message like "The value stored in X[3] is less than the value stored in X[5]."

The second computer-based lesson used in this study which was developed by Boysen (1985) is called MiniPas. The purpose of MiniPas is
to provide the novice programmer with a learning environment which is more transparent than the traditional programming environment. MiniPas is a more transparent learning environment in the sense that the beginner can directly concentrate on the Pascal programming concepts without having to simultaneously deal with a multitude of other details which can interfere with the primary learning task. In the traditional programming environment, the beginner must learn how to use a text editor, a compiler, and possibly a linker. The beginner must also learn several operating system commands in order to use these three pieces of software -- plus the beginner must also learn how to decipher and react to error messages which can be given by the operating system, the editor, the compiler, and the linker. In addition, the novice must learn how to use temporary output statements (WRITELNs) to check the value of variables during the debugging process.

To use MiniPas, only a few operating system commands need to be learned by the student (e.g., how to move, copy, and rename text files). Once the student is in the MiniPas program, the student is guided between modes by menus; i.e., it is menu driven. The options on the main menu include (1) an introduction and overview of MiniPas, (2) a tutorial to learn the text editor commands, (3) the actual MiniPas programming mode, and of course, (4) a way to exit the program. The computer screen as it appears in the Programming Mode is shown in Figure 4.

There are four main areas on the computer screen in the MiniPas programming mode: menu, prompt, program, and memory areas. These four areas are designated in Figure 5. The menu area is used by the student
Figure 4. Layout of the Computer Screen for MiniPas
with Visible Memory

to access the text editor, compile programs, run programs, delete the current program, receive "on line" help, and exit back to the main menu. Because of the programming mode menu, the student does not need to learn the operating system commands which are necessary in the traditional programming environment in order to use the text editor and the compiler and to run the compiled program. All compiler error messages and user prompts appear in the prompt area at the bottom of the screen. The Pascal program which the student is currently working on appears in the program area of the screen. The purpose of the memory area is to show the value of variables used in the program. Additional
Figure 5. The Four Main Areas of the Computer Screen in the MiniPas Programming Mode

KEY

A  Menu Area  
B  Prompt Area  
C  Program Area  
D  Memory Area  

Program  Memory

*Edit Compile Run Delete  help  exit
details about the memory area are provided later in this chapter.

The basic scenario for a student using MiniPas is now given. The student enters the editor to write and modify Pascal programs. When the editing is complete, the program is compiled. If a syntax error is present, the compiler error message is given at the bottom of the screen in the prompt area. To correct the error, the student returns to the editor. Upon entering the editor immediately after encountering a compiler error, the editor places the cursor in the line in which the compiler error was encountered. The student can make any editing changes deemed necessary and then recompile. After a successful compile, the program can be run. The way in which a program executes in MiniPas is the primary reason for using the MiniPas Programming environment.

The MiniPas runtime environment differs from the traditional Pascal runtime environment in two major ways. The first major change has to do with the fact that the original Pascal program appears in the program area and remains there throughout the execution of the program. The computer screen as it appears during runtime is shown in Figure 6. As the program executes, an arrow points to the line in the original program which is currently being executed. After MiniPas completes the execution of the line in the Pascal program, MiniPas waits until the student presses the RETURN key before advancing the arrow to the next line of the Pascal program and executing the next line.

The second major difference between runtime in MiniPas and the traditional runtime environment deals with the memory area on the right side of the MiniPas screen. When MiniPas begins to execute a program,
Program

```pascal
PROGRAM Summation(INPUT, OUTPUT);
VAR
    I, Sum, High : INTEGER;
BEGIN
    WRITELN('Count how high?');
    READLN(High);
    Sum := 0;
    FOR I := 1 TO High DO
        BEGIN
            Sum := Sum + I;
            WRITELN('I = ', I, ', SUM= ', Sum);
        END;
    WRITELN('The final sum is ', Sum);
END.
```

<table>
<thead>
<tr>
<th>Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>HIGH</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Layout of the Computer Screen For Visible Memory MiniPas While a Program is Executing

the first task MiniPas does after displaying the original Pascal program and after displaying the flow of execution arrow (which was discussed in the above paragraph) is to display all variables declared in the Pascal program. The variables are displayed as a stack of boxes which represent the actual variable memory locations. To the left of each box is the name of each variable as it was declared by the student in the Pascal program. As MiniPas executes a line of the Pascal program, if the line was a statement (e.g., READ, READLN, or assignment statement) which results in a change of the contents of any memory location,
MiniPas makes the corresponding change in the appropriate box in the memory area on the screen. When the contents of a box in the memory area is made, MiniPas flashes the respective box to draw the attention of the student to the memory area.

A few minor differences between runtime in MiniPas and the traditional runtime environment in addition to the two major differences already discussed are also noteworthy. If the Pascal program is larger than what fits in the program area, MiniPas automatically replots the program area of the screen so that the Pascal line which is currently being executed does appear in the program area on the screen. Since the program area, memory area, and menu area occupies over 90% of the screen's total area, MiniPas cannot use the screen during runtime in the same way as the traditional runtime screen is handled. MiniPas is restricted to using only the prompt area on the screen for input from the user (READs and READLNs) and output to the user (WRITE and WRITELN).

Several researchers have reported that beginning programming students have difficulty understanding the flow of control of a program and the changes in the various memory locations as the program executes (Barr, Beard, and Atkinson, 1976; Pea, 1986). MiniPas was designed to provide the novice programmer with a means of tracing the flow of control during runtime and to visually see the changes to memory as the changes occur during runtime. In other words, MiniPas was designed to provide the novice programmer with a "window" to see what was actually happening inside of the computer. DuBoulay, O'Shea, and Monk (1981) contend that programming students must "see the works" or what they call
the glass box approach to learning computer programming. If the internal workings of the computer are hidden from the user (called the black box approach), the students do not have a concrete model or a conceptual basis upon which to organize and assimilate the new programming concepts which are being learned.

Two versions of MiniPas were used in this experiment. The first version, called visible memory MiniPas, is as per the above description. The layout of the computer screen for visible memory is shown in Figure 4. The second version, called hidden memory MiniPas, is identical to the first version, but the memory area of the screen is not shown. The layout of the computer screen for hidden memory operations is shown in Figure 7.

Research Design

In an attempt to determine the effects of the independent variables on the dependent variable, an experiment was conducted using the model shown in Figure 8. The two independent variables for this study were: (1) the manipulative model, a nominal variable with two categories (MemOps, students who had the MemOps experience, and Placebo, students who had the placebo problem solving experience); and (2) Memory, a nominal variable with two categories (Visible, students who had visible memory in MiniPas, and Hidden, students who had hidden or invisible memory in MiniPas).
Program

PROGRAM Summation(INPUT, OUTPUT);
VAR
I, Sum, High : INTEGER;
BEGIN
WRITELN('Count how high?');
READLN(High);
Sum := 0;
FOR I := 1 TO High DO
BEGIN
  Sum := Sum + I;
  WRITELN('I=', I, ', ', Sum);
END;
WRITELN('The final sum is ', Sum);
END.

Menu

Edit Compile Run Delete \( \leftarrow \rightarrow \text{ help} \rightarrow \rightarrow \text{ exit} \)

I= 1 Sum= 1

Figure 7. Layout of the Computer Screen for Hidden Memory MiniPas
While a Program is Executing

Manipulative Memory
Model

Subjects

MemOps

Visible

Hidden

Placebo

Visible

Hidden

Figure 8. The Research Model
Research Procedures

On the first day of class (a two hour lab), the course syllabus was distributed and discussed. Then VAX usernames were given to each student and students learned how to log on to VAX, how to access CAS, and were directed to run various CAS lessons. The purpose of this activity was to familiarize the students with the use of the VAX terminal and CAS in preparation for the first experimental treatment, MemOps.

During the next class period (common, one hour lecture), four tasks were accomplished: (1) a rationale for splitting the labs into two groups was given, (2) survey forms were filled out, (3) lab schedules were distributed, and (4) the presentation of course content commenced. The first task was providing the rational for splitting the labs into two groups. Students were told that several computer-based lessons were developed for use in this class. In order to determine if these lessons were doing any good and if they were worthwhile, the class would be split into two groups so that the instructors (this researcher and an associate, Hooper, 1986a) could provide more individual help and attention while they (the students) used the lessons in class. This was all true, but the overriding reason for splitting each lab into two groups was to reduce the possibility of students noticing the visible and hidden memory differences which were obvious when MiniPas was used.

The second task was to fill out the survey forms. Students were told to fill out the survey form to provide background information which would be used in this research project. (A copy of the survey can be
found in Appendix D.) The third task was the distribution of lab schedules. (A copy of the lab schedule can be found in Appendix B.) The fourth and final task was the presentation of regular course content. The remainder of the second class period was used to define terms (character, byte, kilobyte, bit, nibble, RAM, ROM, etc.) which beginning programming students would be encountering.

Since this researcher anticipated student questions which would have to be answered according to membership in an experimental group, students were placed into one of four color groups (gold, green, red, blue). Each color was a code name for one of the four experimental groups. The four color codes which were assigned to the four experimental treatment groups are listed in Figure 9. When the lab schedules were distributed, students were told that during the next lab session they would be assigned to one of four color groups. By looking up their respective color group on the lab schedule, the students could determine where they had to go for lab sessions for the first half of the semester.

<table>
<thead>
<tr>
<th>Experimental Group</th>
<th>Color Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemOps &amp; Visible</td>
<td>Gold</td>
</tr>
<tr>
<td>MemOps &amp; Hidden</td>
<td>Blue</td>
</tr>
<tr>
<td>Placebo &amp; Visible</td>
<td>Red</td>
</tr>
<tr>
<td>Placebo &amp; Hidden</td>
<td>Green</td>
</tr>
</tbody>
</table>

Figure 9. Color Codes which were assigned to the Four Experimental Treatment Groups
The data collected by the survey were used to classify each student into one of three groups: (1) no computer experience, (2) an introductory experience in BASIC programming or wordprocessing, and (3) programming experience in Pascal or FORTRAN. Students who already knew how to program or who have had various levels computer related experiences may already possess the understanding or at least have some type of mental model of how computer memory operates; therefore, it was essential that the students were assigned to the experimental groups using a stratified, random assignment technique.

Half of the students in each of the three stratified groups were randomly assigned to the MemOps group, and the other half were assigned to the placebo group. Then within each of these six groups, half of the subjects were randomly assigned to the Visible Memory group and the remaining subjects were placed into the Hidden Memory group. See Table 1 for a breakdown of the experimental groupings.

At the beginning of the third lab, students received VAX usernames which they were to use during subsequent labs. Written on the username cards was the color designating which group the student belonged. The remaining part of the third class meeting was used for the manipulative model treatment, MemOps or the placebo.

For all labs after the third class session, students went to one of the two respective lab sites according to their group color. The first hour of each lab was dedicated to the presentation of various programming concepts plus discussion and questions by the students were encouraged. During the second hour of the lab, students worked in the computer lab on various programming problems and exercises related
Table 1. Number of Subjects According to Level of Computer Experience and Type of Treatment Received

<table>
<thead>
<tr>
<th>Class Section</th>
<th>Level of Computer Experience</th>
<th>Treatment Groups</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Visible Memory</td>
<td>Hidden Memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MemOps</td>
<td>Placebo</td>
<td>MemOps</td>
</tr>
<tr>
<td>A</td>
<td>None</td>
<td>1 3&lt;sup&gt;ab&lt;/sup&gt;</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intro to BASIC/word processing</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pascal and/or FORTRAN</td>
<td>4 1</td>
<td>2 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>6 6</td>
<td>5 5</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>None</td>
<td>3 2</td>
<td>2&lt;sup&gt;c&lt;/sup&gt; 3&lt;sup&gt;a&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intro to BASIC/word processing</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pascal and/or FORTRAN</td>
<td>2 3&lt;sup&gt;a&lt;/sup&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>5 6</td>
<td>4 6</td>
<td></td>
</tr>
</tbody>
</table>

**Totals for Both Sections**

<table>
<thead>
<tr>
<th></th>
<th>MemOps</th>
<th>Placebo</th>
<th>MemOps</th>
<th>Placebo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Enrollment</td>
<td>11</td>
<td>12</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>For Midterm Exam</td>
<td>11</td>
<td>11</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>For Final Exam</td>
<td>11</td>
<td>9</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 (n=45)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 (n=43)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 (n=38)</td>
<td></td>
</tr>
</tbody>
</table>

<sup>a</sup>Within the first two weeks of class, two students from the designated cell dropped the course.

<sup>b</sup>One student in this cell dropped the class during the second half of the semester.

<sup>c</sup>One student from each of the four marked cells did not take the final exam.
to the current presentation topics.

Since each lab section was split into two groups and each group met concurrently, two presentations had to be made at the same time at two separate lab sites. To minimize the differences between the two concurrent presentations, the researcher prepared detailed lecture notes and overhead transparencies. Before each lab session, the presenters met to discuss the lecture notes and the overhead transparencies to further minimize the differences between the lectures to be given. After each lab presentation, the two presenters "compared notes" to verify that the same amount of material was covered and to determine if any questions asked by the students which resulted in material being presented which was not covered in the lecture notes or overhead transparencies. If any differences occurred, adjustments were made accordingly in the next lab presentation.

Since an overt effort was made to minimize differences between the two concurrent presentations, students essentially received the same material -- regardless of their membership in a particular experimental group. In other words, the lab lectures and exercises were not tailored to capitalize on explaining certain concepts in terms of the visible memory in MiniPas. If a student did, however, ask a question in terms of the visible memory, the presenter would answer the question without drawing any more attention to the visible memory in MiniPas than necessary.

During the eighth lab period of the semester (twelfth class period including the common lectures), the first experimental measurement in the form of an inclass examination was given. The first half of the
examination consisted of short answer and multiple choice items. For the second half of the examination, the students used MiniPas to write two Pascal programs to solve two novel programming tasks: a swap and a simple sort. (See Appendices E and F, respectively, for copies of these two programming problems.) Data for the programming tasks were automatically collected by MiniPas and placed in a history file.

After the midterm break, the lab sections were no longer split. The visible memory and hidden memory groups met as one section. Since MiniPas was not going to be used on a regular basis any more, there was no need to meet in separate groups. Wordprocessing was the first topic covered after midterm break followed by learning how to do Pascal programming using a "regular" compiler on a microcomputer. The last experimental measurement was taken during the last lab of the semester. The last measurement consisted of solving three more novel programming tasks: two were "paper and pencil" solutions and the third was done using MiniPas. (See Appendices G, H, and I, respectively, for copies of these three programming problems.)

Data Collection Procedures

Two separate computer-based lessons were employed in this research study: MemOps and MiniPas. Both of these lessons were written in DAL. To run these lessons, a student had to use a special terminal (GIGI or VT241) which was connected to the VAX minicomputer. A description of MemOps and MiniPas, as viewed from the student's perspective is provided in a previous section of this chapter. A description from the research perspective is given below.
Incorporated into the MemOps lesson was a mechanism to collect the responses and actions of the students as they progressed through the lesson. These responses and actions were placed into text files, called history files, which were later retrieved and analyzed by this researcher. An example of a MemOps history file can be found in Appendix J.

As students worked in the MiniPas programming environment, MiniPas automatically maintained a history file containing the text of the Pascal programs which the students were writing. Every time the student compiled a program which contained a syntax error, a copy of the compiler error message, the row and column in the source code where the compiler encountered the error, and a copy of the program being compiled was placed into the history file. Nothing was placed in the history file for successful compiles. Every time the student would run his or her program, a copy of the Pascal program was placed into the history file. All user input to the program (i.e., user responses to READs and READLNs) were also placed in the history file. These history files were gleaned by this researcher to detect the type of errors made by students and the various algorithms and approaches used. Additional details on how the history files were examined are explained in a subsequent section of this chapter.

Description of the Measures Employed

Measurement data were collected from two different sources: history files and inclass examinations. The history files are described in the previous section. The inclass examinations are discussed below.
The second source of data was the regular inclass examinations. These examinations consisted of short answer (open-ended) and multiple choice items. The examinations were written by the instructor. After the students completed each exam, the multiple choice items were either computer scored or scored by the researcher. All multiple choice items were scored dichotomously (1 for correct, 0 for incorrect). For open-ended questions, one or more criterion were established for each item and the same dichotomous scoring system was employed.

The Kuder-Richardson Formula #20 (KR##20) was used to estimate the reliability of the exam. In order to improve the test reliability, certain test items were discarded. Criterion for item elimination was threefold. First, if the item variance was zero (i.e., either all the students got the item correct or all students got the item incorrect), the respective item was eliminated. Second, point biserial correlations (each item correlated with the total test score) were calculated. An item with point biserial correlation which was negative or less than 0.20 was eliminated. After an item was eliminated, the point biserial correlations and the KR##20 were recalculated based on the remaining items. If a remaining item had a point biserial correlation which was negative or less than 0.20, the respective item was eliminated and the point biserial correlations and the KR##20 were again recalculated based on the remaining items. This process of eliminating items and recalculating the point biserial correlations and the KR##20 was repeated until all items with point biserial correlations which were negative or less than 0.20 were eliminated. The third and final criterion for eliminating test items was based on inter-item correlations (phi
coefficient). If the majority of the phi coefficients for a given item were negative, the respective item was eliminated and the KR#20 was recalculated. If the newly calculated KR#20 dropped, the item was not eliminated. If the newly calculated KR#20 increased, the item was permanently eliminated (Miller, 1986).

After eliminating test items according to the above stated criterion, 24 items remained. The KR#20 estimate of the test reliability was 0.8253. The mean test score was 20.50 and the standard deviation was 3.4799. The lowest score was 6 and two students had a perfect score of 24. A copy of the 24 test items can be found in Appendix P. The inter-item correlations and the point biserial correlations for the 24 test items are provided in Appendix Q.

A Priori Statistical Analyses

A two way analysis of variance (ANOVA) from a multiple regression framework was conducted to examine the first three statistical hypotheses. The General Linear Model (GLM) procedure in the Statistical Analysis System (SAS) was used for calculating the regression model. The two independent, experimental treatment variables were dummy coded as shown in Table 2. There was no need to create an interaction vector of dummy codes since interaction vectors can be designated in the GLM model specification statement in SAS by using the 'X' operator.

The Fisher's Exact Test was conducted to examine the last two statistical hypotheses. This statistical procedure was selected instead of the chi-square test of independence and the t-test for testing the difference between two proportions for two reasons. The first reason
Table 2. Dummy Codes for the Experimental Treatment Variables

<table>
<thead>
<tr>
<th>Manipulative Model Group</th>
<th>Memory Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemOps &amp; Visible</td>
<td>1</td>
</tr>
<tr>
<td>MemOps &amp; Hidden</td>
<td>1</td>
</tr>
<tr>
<td>Placebo &amp; Visible</td>
<td>0</td>
</tr>
<tr>
<td>Placebo &amp; Hidden</td>
<td>0</td>
</tr>
</tbody>
</table>

was due to low cell counts (less than 5) in the 2 by 2 tables. Both the Chi square test and the t-test for proportions require that all four cells in the 2 by 2 table have expected counts of 5 or more. The second reason for selecting the Fisher's Exact Test was based on the fact that the Fisher's Exact Test does not assume that the sample is normally distributed. Refer to Appendix 0 for the formula and procedures used in the Fisher's Exact Test calculations.

An alpha level of p < .05 was used for the testing of all the a priori hypotheses in this study.

Post Hoc Statistical Analyses

Since the background experiences of the students participating in this study had a potential effect on the results of the original regression analysis, six variables were added to the original regression model for the expressed purpose of statistically controlling for the background experiences of the students. The six control variables used in the post hoc analyses were obtained from the survey which the students filled out during the first common lecture period of the semester. (A copy of the survey is provided in Appendix D.) The six
control variables are (1) the amount of past computer-related experience, (2) the number of semester hours of college calculus, (3) the number of semester hours of credit being taken, (4) the number of hours worked per week, (5) the letter grade which the student expected to receive in the IEDT 216 course and (6) an attitude scale rating.

The first of the six control variables is the amount of past computer-related experience (EXP). On the survey, students were asked to list all the computer-related coursework and experiences which they had both in high school and in college. Based on the survey responses, each student was classified into one of three groups: (1) no computer experience, (2) some computer experience including computer literacy, BASIC programming, word processing, and data entry, and (3) Pascal and FORTRAN programming. The distinction between the last two groups was based upon the assumption that students who had Pascal or FORTRAN probably had a formal course taught by an instructor who used either language in some professional capacity. A parallel assumption was made that students who had BASIC programming or a computer literacy course were probably taught by an instructor who did not have a formal programming background and therefore the coursework was not as structured or intensive as a Pascal or FORTRAN course. These two assumptions were supported by the fact that the courses listed for Pascal or FORTRAN were either college level computer science or college level freshman engineering courses and essentially all of the courses listed as BASIC or "computer literacy" courses were high school exploratory courses. A dummy coding scheme as shown in Table 3 was used for entering the computer related experiences into the regression analyses.
Table 3. Dummy Codes for Computer Related Experiences Variables

<table>
<thead>
<tr>
<th>Level of Experience</th>
<th>Variable Names of Dummy Vectors Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NOEXP</td>
</tr>
<tr>
<td>No experience</td>
<td>1</td>
</tr>
<tr>
<td>Some Experience</td>
<td>0</td>
</tr>
<tr>
<td>Pascal or FORTRAN</td>
<td>0</td>
</tr>
</tbody>
</table>

The second of the six control variables is the number of semester hours of college calculus (CALC). On the survey form, a list of commonly taken high school and college mathematics courses was provided. Students were asked to provide the number of semester hours of each that they had taken and passed. The value used in the regression analysis was a ratio variable representing the number of semester hours of college calculus which they had taken and passed. There were three reasons for using the number of semester of hours of college calculus and ignoring the high school mathematics taken and ignoring the other non-calculus college mathematics taken: (1) the reliability of the number of semester hours of high school mathematics coursework was assumed to be low due to the fact that variability between high school mathematics courses was assumed to be high; (2) it was assumed if a student had a solid high school mathematics background, s/he would be taking calculus courses at the college level; a student who had a limited high school mathematics background would have to take algebra, trigonometry, and descriptive geometry at the college level before taking any college calculus; and (3) over 80% of the college mathematic courses listed were ISU calculus courses. Based on these reasons, it
was assumed that using the number of semester hours of college calculus provided a more valid and more reliable measure of mathematics ability than would be provided by including the number of semester hours of high school mathematics and non-calculus college mathematics.

The third and fourth control variables dealt with the amount of time the student had committed to various activities each week. The number of semester hours of credit that the student was carrying during spring semester of 1986 was the third control variable (CREDIT). The fourth control variable (WORK) was the number of hours per week which the student had to work or had committed to other regular weekly activities (e.g., club officer, member of the government of the student body). Both these variables represent ratio data.

The fifth control variable was the letter grade (GRADE) the student expected to receive in IEDT 216. GRADE is an interval variable with an letter grade of an A numerically represented as a 1, A- as a 2, ..., F as a 12.

The sixth and final control variable is an attitude scaling rating (ATT). The attitude scale was included as part of the survey form filled out by the students at the beginning of the semester. ATT consisted of five statements which the students had to respond on a nine point Likert scale. A response of a nine on the first three items indicated a strong agreement for the need and use of computers in college and in the student's future job in industry. On the last two items, the statement and scale were such that a response of one indicated a strong pro computer usage attitude. The final attitude scale was the sum of the first three attitude responses plus the
responses of each of the last two items after subtracting them from ten. Subtracting the last two items from ten before adding them into the sum resulted in five "pro" computer scores being summed together.

The development of a computer usage attitude scale was not one of the purposes of this research study. The attitude scale was developed and included just as a point of curiosity by this researcher. Since the development of the attitude scale was not one of the purposes of this study, no validity or reliability estimates of the attitude scale were determined.

Custom Computer Programs for Data Analysis

Two computer programs were written by this researcher to assist in the analysis of the history files generated by the MemOps and the MiniPas lessons. A third program was also written for performing the Fisher Exact Test calculations. A summary of each program is given below.

Closure Program

The Closure Program was used to analyze the MemOps history files. Every move and comparison which a student made while working through MemOps was recorded in a history file for each student. (See Appendix J for an example of a MemOps history file.) In particular, when the students were doing the ascending or descending sort with hidden memory, did the students do enough comparisons to know the actual order of the array elements or were they just guessing? The Closure Program answered this question and thus enhanced the scoring reliability of the MemOps
history files.

The basic algorithm for the Closure Program was taken from *Data Structures and Algorithms* by Aho, Hopcroft, and Ullman (1983). The algorithm is founded in mathematical graph theory. A description of the algorithm and the data structures used plus the enhancements required for this particular research application are provided in Appendix L. A listing of the Turbo Pascal source code for the Closure Program is provided in Appendix M.

**Byte Comparison Program**

As students worked in the MiniPas environment, the MiniPas program maintained a history file containing copies of the compiled programs which contained syntax errors including the error message and the row and column location of where the compiler encountered the syntax error. Nothing was recorded in the history files for successful compiles. Everytime the student ran a program, a copy of the program along with all user inputs to the program were stored in the history file. The order in which these items were placed in the history file was in chronological order. See Appendix K for an example of a MiniPas history file.

In order to trace the changes which the students made in successive programs stored in the history files, a character by character comparison had to be made between adjacent programs. Manually doing a character by character comparison and manually counting the lines and the columns to locate where the compiler encountered the syntax error was a tedious, time-consuming, and error prone activity.
The Byte Comparison Program was written and implemented to reliably performed these tasks. Additional technical details about the implementation of this program can be found in Appendix N. A listing of the Turbo Pascal source code for the Byte Comparison Program is provided in Appendix N.

**Fisher's Exact Test Program**

The Fisher's Exact Test can be calculated by using Statistics 1 option of the Crosstabs procedure in SPSSX. The Statistics 1 option, however, will only be calculated if the number of cases is less than twenty. Therefore, a microcomputer program was written to calculate this statistic when more than twenty cases were involved. Additional details concerning the design of this microcomputer program and a program listing are provided in Appendix O.

**Analyzing Programming Protocols**

As students worked in the MiniPas programming environment, MiniPas automatically maintained a history file containing all the programming protocols used by the students. In analyzing the programming protocols contained in the history files, this researcher focused on the type of programming practices which the students used which suggested programming practices not typically used by experienced or expert programmers. Special attention was paid to how the students used variables to store data in their programs. The behaviors which were observed were classified in an attempt determine if the experimental treatments affected these behaviors.
CHAPTER IV. FINDINGS

The problem of this study was to determine what effects the use of MemOps and the use of visible memory in MiniPas had on the learning of computer programming by novices. The criterion variables consisted of inclass examination scores, whether or not students obtained workable solutions to five separate novel programming tasks, and programming protocols. For the hypotheses having the inclass examination score as the criterion variable, the data were analyzed using multiple linear regression. The nominal data which indicated whether or not students obtained a workable solution to the novel programming tasks were analyzed with the Fisher's Exact procedure. And the programming protocols were manually gleaned to detect student programming tendencies.

The findings which are reported in this chapter are organized into four parts:

1. Description of Findings - Part One. This part contains the findings of the regression analysis which was used to test the null hypotheses for Research Questions I, II, and III. The dependent variable for each of these was the inclass examination score.

2. Description of Findings - Part Two. This part contains the findings of the Fisher's Exact Test which was used to test the null hypotheses corresponding to the fourth and fifth research questions. The criterion variables for these two research questions were whether or not students obtained workable solutions to five novel programming tasks.

3. Description of Findings - Part Three. The findings for the sixth and final research question are presented in Part Three. Research Question VI examined the type of programming practices which novice programming students employ.

4. Description of Findings - Part Four. Several exploratory post hoc
statistical analyses were conducted and the findings of these analyses are reported in this part.

Description of Findings - Part One

The findings for the first three research questions are reported below. For Research Questions I and II, the two independent variables were (1) the use or non-use of the manipulative model (Mem Ops) and (2) the visibility or non-visibility of the memory area when using MiniPas. Research Question III pertains to the interaction effect between the two independent variables. The criterion variable was inclass examination scores.

The research design was a two by two factorial design. Regression analysis was used to determine the significance of the two main effects and the interaction effect. The number of students in each of the four cells was 11 except for the Mem Ops-Hidden group which contained 9 students. At the onset of the experiment, the cell frequencies were proportional. However, due to students dropping the course, the cell counts became unbalanced. Since the sample size was already small, observations were not thrown out to balance the cell counts. Therefore, the assumption of independence of effects was violated and the findings reported in Part One must be interpreted in light of this violation.

Research Question I

The first research question as stated in Chapter I was: Will there be a difference in learning between students using the manipulative model and students who do not use the manipulative model?

The overall mean of the inclass examination scores for the
students in the MemOps group was higher than the overall mean for the students in the Placebo group. The mean for the MemOps group was 21.50 and the mean for the Placebo group was 19.59. The variance for the MemOps group was 5.00 which was less than the variance for the Placebo group, 17.30. The regression analysis of the data relating to Research Question I did not substantiate the difference between the means of the MemOps group and the Placebo group ($F (1,38) = 1.73, p < .196$). The statistical null hypothesis of Research Question I, therefore, was not rejected. In other words, there was no statistically significant difference in the means of the examination scores between the students using the manipulative model before language instruction and the students who did not use the manipulative model. These data are shown in Tables 4 and 5.

**Research Question II**

The second research question as stated in Chapter I was: Will there be a difference in learning between students using visible memory and students who use the invisible memory?

The mean of the inclass examination scores for the Visible group was lower than the mean for the Hidden group. The means were 20.36 and 20.65. The variance for the Visible group was greater than the variance for the Hidden group, 16.53 and 7.82. The regression analysis of the data relating to Research Question II did not substantiate the difference between the means of the Hidden group and the Visible group ($F (1,38) = 0.03, p < .855$). The statistical null hypothesis of Research Question II, therefore, was not rejected. In other words,
Table 4. Summary of Regression Analysis of Inclass Examinations

<table>
<thead>
<tr>
<th>Source</th>
<th>df</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>p</th>
<th>R^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>3</td>
<td>39.85</td>
<td>13.28</td>
<td>1.11</td>
<td>.36</td>
<td>.08</td>
</tr>
<tr>
<td>Error</td>
<td>38</td>
<td>456.65</td>
<td>12.02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>41</td>
<td>496.50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partial Sums of Squares^a

<table>
<thead>
<tr>
<th>Source</th>
<th>df</th>
<th>SS</th>
<th>partial F</th>
<th>p</th>
<th>partial R^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemOps-A</td>
<td>1</td>
<td>20.81</td>
<td>1.73</td>
<td>.196</td>
<td>.04</td>
</tr>
<tr>
<td>Memory-B</td>
<td>1</td>
<td>0.41</td>
<td>0.03</td>
<td>.855</td>
<td>.00</td>
</tr>
<tr>
<td>A * B</td>
<td>1</td>
<td>0.14</td>
<td>0.01</td>
<td>.914</td>
<td>.00</td>
</tr>
</tbody>
</table>

^aType III Sums of Squares.

Table 5. Summary of Inclass Examination Scores by Manipulative Model Treatment Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
<th>Variance</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Placebo</td>
<td>22</td>
<td>19.59</td>
<td>4.16</td>
<td>17.30</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>MemOps</td>
<td>20</td>
<td>21.50</td>
<td>2.24</td>
<td>5.00</td>
<td>16</td>
<td>24</td>
</tr>
</tbody>
</table>
there was no statistically significant difference in the means of the examination scores between the students using visible memory in MiniPas and the students using hidden memory in MiniPas. These data are shown in Tables 4 and 6.

Table 6. Summary of Inclass Examination Scores by Memory Treatment Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
<th>Variance</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden</td>
<td>20</td>
<td>20.65</td>
<td>2.80</td>
<td>7.82</td>
<td>14</td>
<td>23</td>
</tr>
<tr>
<td>Visible</td>
<td>22</td>
<td>20.36</td>
<td>4.07</td>
<td>16.53</td>
<td>6</td>
<td>24</td>
</tr>
</tbody>
</table>

Research Question III

The third research question as stated in Chapter I was: Will there be an interaction effect between student membership in the Manipulative Model groups (MemOps or placebo groups) and student membership in the Memory groups (visible or hidden groups)?

The means of the inclass examination scores for the four experimental groups were 21.27 for the MemOps-Visible group, 21.78 for the MemOps-Hidden group, 19.45 for the Placebo-Visible group, and 19.73 for the Placebo-Hidden group. The variances for the four groups were 7.02, 2.94, 25.87, and 10.42, respectively. A regression analysis of the data relating to Research Question III did not substantiate the interaction effect due to student membership in a particular Manipulative Model group and student membership in a particular Memory group ($F (1,38) = 0.01, p < .914$). These data are provided in Tables 4 and 7.
Table 7. Summary of Inclass Examination Scores by Manipulative Model and Memory Treatment Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
<th>Variance</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemOps-Visible</td>
<td>11</td>
<td>21.27</td>
<td>2.65</td>
<td>7.02</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>MemOps-Hidden</td>
<td>9</td>
<td>21.78</td>
<td>1.72</td>
<td>2.94</td>
<td>18</td>
<td>23</td>
</tr>
<tr>
<td>Placebo-Visible</td>
<td>11</td>
<td>19.45</td>
<td>5.09</td>
<td>25.87</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>Placebo-Hidden</td>
<td>11</td>
<td>19.73</td>
<td>3.23</td>
<td>10.42</td>
<td>14</td>
<td>23</td>
</tr>
</tbody>
</table>

Description of Findings - Part Two

Reported in this section are the findings which correspond to Research Questions IV and V. For Research Question IV, the independent variable was the use or non-use of the manipulative model (MemOps). The independent variable for the Research Question V was the use of visible or hidden memory when using MiniPas. The five criterion variables were whether or not students obtained workable solutions to five novel programming tasks. These nominal data were analyzed using the Fisher's Exact procedure.

The five novel programming tasks are listed and summarized below (the appendices referenced inside the parentheses contain a copy of the respective novel programming task as posed to the student):

1. Swap - input any two numeric values from the user, interchange the values of the two variables, and then output the two values (Appendix E);

2. Simple Sort - input any three numeric values from the user and then output the three values in ascending order (Appendix F);

3. Reverse - given a one-dimensional array which contains numeric values, provide the Pascal code to reverse the order of the values
in the array (Appendix H);

4. Compare - given two one-dimensional arrays which contain numeric values, provide the Pascal code to compare the values in the corresponding slots of each array and for each pair of slots, output which array contains the larger value (Appendix G);

5. Array Sort - given a one-dimensional array which contains numeric values, sort the values stored in the array into ascending order (Appendix I).

Research Question IV

The fourth research question as stated in Chapter I was: Will there be a difference between the proportion of MemOps students who obtain workable solutions to novel programming tasks and the proportion of Placebo students who obtain workable solutions?

A summary of the frequency counts of students who obtained workable solutions and students who did not obtain workable solutions to the five novel programming tasks is shown in Table 8. For the Swap programming task, 15 out of 20 MemOps students obtained a workable solution compared to 8 of 23 students in the Placebo group (Fisher's Exact $p < .009$). Therefore, the null hypothesis for the Swap programming task was rejected in favor of the alternative hypothesis, there was a difference between the proportion of MemOps students and the proportion of Placebo students who obtained workable solutions.

The frequency counts for each of the four remaining novel programming tasks were not statistically significant at the .05 alpha level. Therefore, the four null hypotheses of independence for the Simple Sort, Reverse, Compare, and Array Sort programming tasks were not rejected.
Table 8. Summary of Obtaining Workable Solutions to Novel Programming Tasks by Manipulative Model Treatment Groups

<table>
<thead>
<tr>
<th>Novel Task</th>
<th>Obtained Solution</th>
<th>Did Not Obtain Solution</th>
<th>Fisher's Exact p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MemOps</td>
<td>Placebo</td>
<td>MemOps</td>
</tr>
<tr>
<td>Swap</td>
<td>43</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>Simple Sort</td>
<td>43</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Reverse</td>
<td>38</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Compare</td>
<td>38</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>Array Sort</td>
<td>38</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

*p < .05.

Research Question V

The fifth research question as stated in Chapter I was: Will there be a difference between the proportion of visible memory students who obtain workable solutions to novel programming tasks and the proportion of hidden memory students who obtain workable solutions?

A summary of the frequency counts of students who obtained workable solutions and students who did not obtain workable solutions to the five novel programming tasks is shown in Table 9. The proportion of visible memory students who obtained workable solutions compared to the proportion of hidden memory students who obtained workable solutions on each of the five novel programming tasks were nearly identical. For example, 52% (10 out of 23) of the visible memory students obtained workable solutions to the Swap programming task as compared to 50% (10 out of 20) of the hidden memory students who obtained workable solutions. The results of the Fisher's Exact procedure indicated that the difference between these two proportions was not statistically significant (Fisher's Exact p < .673).
Since the proportion of visible memory students and the proportion of hidden memory students who obtained workable solutions to all five programming problems were nearly identical, the differences between the five pairs of proportions were not significant. Therefore, the five null hypotheses of independence between the use of visible and hidden memory in MiniPas and obtaining workable solutions to Swap, Simple Sort, Reverse, Compare, and Array Sort programming tasks were not rejected.

Table 9. Summary of Obtaining Workable Solutions to Novel Programming Tasks by Memory Treatment Groups

<table>
<thead>
<tr>
<th>Novel Task</th>
<th>Obtained Solution</th>
<th>Did Not Obtain Solution</th>
<th>Fisher's Exact p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Visible</td>
<td>Hidden</td>
<td>Visible</td>
</tr>
<tr>
<td>Swap</td>
<td>43</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>Simple Sort</td>
<td>43</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Reverse</td>
<td>38</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Compare</td>
<td>38</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Array Sort</td>
<td>38</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Description of Findings - Part Three

The findings for the sixth research question are reported in this part. This research question is descriptive in nature rather than inferential; thus, there is no statistical hypothesis for this research question. Instead, anecdotes of the various programming practices which may suggest programming practices that are not typically used by experienced or expert programmers are provided. The data for this research question were obtained from the history files, the "paper and pencil" solutions, and from the daily lab exercises.

The organization of Part Three is, first, to restate the research
question and, second, to provide the anecdotes. The anecdotes are enumerated Anecdote 1, Anecdote 2, Anecdote 3, etc. Figures containing the Pascal code which exemplified the programming practice are also provided. Ellipses points are used in the Pascal listings to designate omissions which are not pertinent to the respective anecdote being discussed. The physical layout of the Pascal listings (indentation, upper and lower case letters, etc.) was altered to increase the readability and consistency of the figures.

Research Question VI

The sixth research question as stated in Chapter I was: What types of programming practices do beginning programmers use that may suggest programming practices which are not typically used by experienced or expert programmers?

Anecdotes

The anecdotes of the various programming practices as gleaned from the MiniPas history files, the "paper and pencil" tests, and the daily programming assignments are enumerated in this section. The programming practices were not selected based on frequency counts but were selected based on their deviation from expected programming practice. Expected programming practice was defined as those programming techniques which were typically used by experienced or expert programmers. The programming practices cited were not isolated instances. All except anecdotes four and seven were employed by six or more students. The fourth and seventh anecdote were found in only two solutions, but the
Anecdote 1  Several students used the FOR loop but failed to take advantage of the index within the FOR loop body. Instead of using the index, the students set up a second integer variable which was separate from the index. This integer variable was initialized to zero just prior to the FOR loop. The first statement in the body of the FOR loop was an assignment statement which incremented the second integer variable by one. The value of the second integer variable and the value of the FOR loop index were always identical during the execution of the remaining part of the FOR loop body. The protocol for this anecdote is provided in Figure 10.

```plaintext
CONST
  Max = 7;
VAR
  I, J : INTEGER;
BEGIN
  J := 0;
  FOR I := 1 TO Max DO
      BEGIN
          J := J + 1;
      END;
END;
```

Figure 10. Protocol A

Anecdote 2  When doing the Compare programming task, over one third of the students employed unique variable identifiers for indexes to FOR loops which were independent of each other. FOR loops which are not nested inside each other, do not require the use of unique index
variable identifiers. Whether or not these students realized that the control variables were independent of each other cannot be ascertained from the protocols. The Pascal listing for this anecdote is provided in Figure 11.

```
CONST
  Max = 7;
VAR
  I, J, K : INTEGER;
  X, Y : ARRAY[1..Max] OF INTEGER;
BEGIN
  FOR I := 1 TO Max DO 
    READLN(X[I]);
  FOR J := 1 TO Max DO 
    READLN(Y[J]);
  FOR K := 1 TO Max DO 
    BEGIN
      ...
      END;
      ...
```

Figure 11. Protocol B

**Anecdote 3** In doing the Compare programming problem, several students' solutions would have been correct if both FOR loop control variables incremented concurrently instead of independently. These students did not realize that at the end of a given FOR loop body, the respective control variable was automatically incremented and then the test was made to determine whether or not the body of the FOR loop was to be executed again. If a second FOR loop was nested inside of the first FOR loop, the incrementing of the respective control variables are independent of each other, not concurrent. What the students' solutions did, instead of comparing the corresponding values in the two arrays,
was to compare all the values in one array with each of the values in the other array. Refer to Figure 12 for the programming protocol for this anecdote.

```
CONST
  Max = 7;
VAR
  I, J, K: INTEGER;
  X, Y: ARRAY[1..Max] OF INTEGER;
BEGIN
  ...
  FOR I := 1 TO Max DO
    BEGIN
      FOR J := 1 TO Max DO
        BEGIN
          IF X[I] > Y[J]
            THEN
              ...
```

Figure 12. Protocol C

**Anecdote 4** Two students attempted to implement one FOR loop with two control variables. To make this type of error at the time when the FOR loop was first being learned was understandable. However, to make this type of error ten weeks later after having used the FOR loop in many different applications was unusual. See Figure 13 for the protocol which corresponds to this anecdote.

**Anecdote 5** Several students initialized an integer variable to zero just prior to a FOR loop which used the integer variable as its index. These students did not understand that one of the operations which was automatically performed by the FOR loop was to initialize the control variable to the starting bound. This misconception was observed
CONST
  Max = 7;
VAR
  I, J, K : INTEGER;
  X, Y : ARRAY[1..Max] OF INTEGER;
BEGIN
...
FOR I,J := 1 TO Max DO
BEGIN
  IF X[I] > Y[J]
  THEN ...

Figure 13. Protocol D

on numerous occasions and was not unique to a particular programming problem. See Figure 14 for the respective protocol discussed above.

K := 0;
FOR K := 1 TO Max DO ...

Figure 14. Protocol E

Anecdote 6 On numerous occasions, several students did not capitalize on the capabilities of the WRITELN statement. Instead of placing numeric expressions in the WRITELN parameter list, the students would first assign the numeric expression to a temporary variable and then use the temporary variable in the WRITELN parameter list. A similar behavior was also observed in connection with array values. Several students would first assign the given array location to a temporary variable and then use the temporary variable in the WRITELN parameter list. See Figures 15 and 16 for the two protocols which exemplify this anecdote.
In situations where the array location of interest was a calculated value, students would not use an integer expression as an array subscript, but would instead first calculate the subscript and assign this value to a temporary variable and then use the single variable as the array subscript. This programming behavior was observed on numerous occasions in programs which used arrays but was not unique to a particular programming problem. The two protocols for this anecdote are provided in Figures 17 and 18.

```
J := I + 1;
WRITELN(X[J]);
```

Figure 17. Protocol H
Anecdote 7 In solving the Swap problem, several students thought that a math operator had to be included on the right side of the assignment statement -- even though no math operations were needed. The two protocols for this anecdote are provided in Figures 19 and 20.

Anecdote 8 Periodically, students used an assignment statement which assigned the value of a variable to itself. This behavior occurred more frequently just prior to the assigning of the value of the variable to another variable. This misconception was observed on numerous occasions and was not unique to a particular programming problem. Figures 21 and 22 contain the protocol for this anecdote.
Anecdote 9 Many students' interpretation of compiler error messages was literal only. They never considered the possibility that the error message was symptomatic and that some other error in the program. For example, if there was an extra comma or a missing value in a WRITELN statement, the compiler error message was "')' expected". The students who literally interpreted the error message reacted mechanically by adding a right parenthesis.

Anecdote 10 Some students never read the compiler error messages nor paid attention to where the cursor marked the location in the program where the syntax error was detected. The evidence which suggested that the students never paid attention to the compiler error messages was the nature of the editing changes which were made after receiving the error messages. The editing changes were totally unrelated to the error messages and the location of the editing changes were frequently after the location marked by the compiler.

Anecdote 11 On several occasions, students would make the same syntax error repeatedly in a program. Upon encountering the compiler
error message, the students examined the program to find the error and then fixed all occurrences of the error throughout the program. The program was compiled again. The error was still present. The students reassessed the situation and again changed all occurrences of the error throughout the program and recompiled. The error still was not fixed. This happened several more times before the students fixed only the first occurrence of the error and did not fix the rest of them. It still took several more editing changes and compile attempts before the error was found. Once the first error was corrected, the compiler flagged the second occurrence of the same error. The students made the correction and recompiled. This time the third occurrence of the error was detected. Eventually all the syntax errors in the program were corrected.

Anecdote 12  Many students made the same syntax error repeatedly throughout their program, but failed to recognize in each instance that the same mistake was causing the compiler errors. For example, the variable name "surface area" which contained an illegal blank character was used consistently throughout the program. The first time the compiler detected the improper variable name was in the variable declaration area. The student did not immediately recognize the error and had to edit and recompile several times before he finally eliminated the blank character between "surface" and "area". The student fixed several other syntax errors in the program before the compiler ran across "surface area" in an assignment statement. The student did not immediately recognize the error. After several editing changes and recompiles he finally fixed the error by eliminating the
The behaviors of the students described in the previous anecdote and in this anecdote were similar from the perspective that both groups of students repeatedly made the same mistake through the program. However, the manner in which the students in each group fixed the error was different. When the students in the former group encountered the error, they knew that they had several occurrences of the same error and therefore they fixed all occurrences of the error before recompiling. Whereas, the students in the latter group did not recognize each occurrence of the error as being the same error that they had fixed moments before.

Anecdote 13 There was a considerable amount of variation in the manner in which the students test ran their programs. For some students, if the first test run of their program gave the results which they expected, they would quit working on the program. Programs which did not run as expected were modified until the program did run as expected. Other students would extensively test their programs. These students used various combinations of input data to verify that their program worked with the various data combinations.

Description of Findings - Part Four

In an effort to explore additional effects that the manipulative models may have had on the student behaviors which were not revealed by the a prior analyses, several post hoc analyses were conducted. The findings of the post hoc analyses are reported in this part of the chapter. The findings are organized into three sections: Regression
Models and Problem Solving Approaches. Two different regression models were examined to determine the effect of the use or non-use of MemOps and the effect of visible or hidden memory after adding several combinations of control variables to the original regression model. In the section entitled "Problem Solving Approaches," the various methods which students employed in solving the exercises in MemOps and two the novel programming task were examined.

Regression Models

Several exploratory post hoc statistical analyses were conducted to ascertain additional sources of variance in the examination scores which were not accounted for in the regression model as reported in Part One. Two different post hoc regression analyses were performed: a full model consisting of nine independent variables and a stepwise forward analysis. The full model regression was conducted because other descriptive data were available that were considered potentially good predictors in any subject area. It was felt greater sensitivity to the treatment effects could be achieved by partialling out the effects of the additional descriptive variables. Since an analysis of covariance could not be done due to the small sample size, a stepwise forward analyses was conducted to explore which variables would be the best predictors of success for the inclass examinations.

The Full Model

The original regression model which was reported in Part One contained three independent variables: MemOps, Memory, and the interaction between MemOps and Memory. Six additional variables were added to the original three variable model. The six
variables are (1) the number of semester hours of credit taken during the spring semester of 1986 (CREDIT), (2) the total number of hours per week which the student had to work including volunteer activities, service as a club officer, or any other commitment which the student had on a regular weekly basis (WORK), (3) a dichotomous variable indicating whether or not the student had any prior computer-related experiences (EXP), (4) the number of semester hours of college calculus passed (CALC), (5) the score from an attitude scale was used to determine how important students felt computers would be in their college and professional careers (ATT), and (6) the grade the students expected to receive in IEDT 216 (GRADE). This nine variable regression model is hereafter called the Full Model.

The Full Model was statistically significant ($F(9, 32) = 2.58$, $p < .023$) and accounted for 42% of the variance. Three variables were significant after controlling for the remaining variables in the Full Model: WORK ($F(1, 32) = 4.64$, $p < .039$), EXP ($F(1, 32) = 4.99$, $p < .033$), and CALC ($F(1, 32) = 5.34$, $p < .028$).

A summary of the regression results for the Full Model is shown in Table 10. The means and standard deviations for the six additional variables are shown in Table 11. A correlation matrix for all nine independent variables is shown in Table 12.

The Stepwise Models A stepwise forward regression analysis was conducted for exploratory purposes to determine which variables would be the best predictors of performance on the inclass examination. The RMAX option of the STEPWISE procedure in SAS was used. In summary, the RMAX option first found the best one variable regression model which had the
Table 10. Summary of Regression Analysis of Inclass Examinations Using the Full Model

<table>
<thead>
<tr>
<th>Source</th>
<th>df</th>
<th>SS</th>
<th>MS</th>
<th>F</th>
<th>p</th>
<th>R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>9</td>
<td>208.88</td>
<td>23.21</td>
<td>2.58*</td>
<td>.023</td>
<td>.42</td>
</tr>
<tr>
<td>Error</td>
<td>32</td>
<td>287.62</td>
<td>8.99</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>41</td>
<td>496.50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partial Sums of Squares

<table>
<thead>
<tr>
<th>Source</th>
<th>df</th>
<th>SS</th>
<th>partial F</th>
<th>p</th>
<th>partial R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemOps</td>
<td>1</td>
<td>0.46</td>
<td>0.05</td>
<td>.822</td>
<td>.00</td>
</tr>
<tr>
<td>Visible</td>
<td>1</td>
<td>8.68</td>
<td>0.97</td>
<td>.333</td>
<td>.02</td>
</tr>
<tr>
<td>A * B</td>
<td>1</td>
<td>10.84</td>
<td>1.21</td>
<td>.280</td>
<td>.02</td>
</tr>
<tr>
<td>CREDIT</td>
<td>1</td>
<td>5.83</td>
<td>0.65</td>
<td>.427</td>
<td>.01</td>
</tr>
<tr>
<td>WORK</td>
<td>1</td>
<td>41.71</td>
<td>4.64*</td>
<td>.039</td>
<td>.08</td>
</tr>
<tr>
<td>EXP</td>
<td>1</td>
<td>44.84</td>
<td>4.99*</td>
<td>.033</td>
<td>.09</td>
</tr>
<tr>
<td>CALC</td>
<td>1</td>
<td>47.96</td>
<td>5.34*</td>
<td>.028</td>
<td>.10</td>
</tr>
<tr>
<td>ATT</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>.993</td>
<td>.00</td>
</tr>
<tr>
<td>GRADE</td>
<td>1</td>
<td>21.21</td>
<td>2.36</td>
<td>.134</td>
<td>.04</td>
</tr>
</tbody>
</table>

a Type III Sums of Squares.
*p < .05.

Table 11. Summary of the Six Additional Independent Variables and the Criterion Variable (n = 42)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>SD</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>14.07</td>
<td>11.06</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>EXP</td>
<td>0.69</td>
<td>0.47</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CALC</td>
<td>3.17</td>
<td>5.72</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>CREDIT</td>
<td>14.50</td>
<td>2.28</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>GRADE</td>
<td>3.29</td>
<td>1.76</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>ATT</td>
<td>32.88</td>
<td>4.62</td>
<td>21</td>
<td>43</td>
</tr>
<tr>
<td>SCORE</td>
<td>20.50</td>
<td>3.48</td>
<td>6</td>
<td>24</td>
</tr>
</tbody>
</table>
Table 12. Correlation Matrix (n = 42)

<table>
<thead>
<tr>
<th></th>
<th>MemOps</th>
<th>Memory</th>
<th>A*B</th>
<th>WORK</th>
<th>EXP</th>
<th>CALC</th>
<th>CREDIT</th>
<th>GRADE</th>
<th>ATT</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemOps-A</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory-B</td>
<td>.050</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A*B</td>
<td>.625</td>
<td>.568</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORK</td>
<td>.196</td>
<td>.103</td>
<td>.335</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>.123</td>
<td>-.123</td>
<td>-.070</td>
<td>-.143</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALC</td>
<td>.333</td>
<td>.056</td>
<td>.180</td>
<td>.047</td>
<td>.200</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREDIT</td>
<td>-.021</td>
<td>-.276</td>
<td>-.229</td>
<td>-.384</td>
<td>.309</td>
<td>-.111</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRADE</td>
<td>-.157</td>
<td>.020</td>
<td>-.129</td>
<td>-.089</td>
<td>-.097</td>
<td>-.381</td>
<td>.073</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATT</td>
<td>.234</td>
<td>.027</td>
<td>.276</td>
<td>.034</td>
<td>.073</td>
<td>.078</td>
<td>-.344</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCORE</td>
<td>.277</td>
<td>-.042</td>
<td>.134</td>
<td>-.253</td>
<td>-.397</td>
<td>.395</td>
<td>.067</td>
<td>.040</td>
<td>.051</td>
<td>1.000</td>
</tr>
</tbody>
</table>

The highest R squared, then the best possible two variable model was found, and so on until all the variables had been included. The order in which the nine variables entered into the successive stepwise models was EXP, CALC, WORK, GRADE, A * B (interaction between the two main effects), Memory-B, CREDIT, MemOps-A, and ATT.

The first stepwise model and, therefore, the eight remaining stepwise regression models were significant at the .05 alpha level. The single variable which accounted for the largest amount of the variance in the examination scores was EXP; the amount of variance accounted for was 16%. The additional amount of variance accounted for by the addition of CALC to the one variable model containing EXP was statistically significant. The two variable model containing EXP and CALC accounted for 26% of the variance. Entering the third variable, WORK, into the model containing EXP and CALC did not increase the model sums of squares a significant amount. A summary of the stepwise forward regression results is shown in Table 13.
Table 13. Summary of the Stepwise Forward Regression Results for Inclass Examination Scores

<table>
<thead>
<tr>
<th>Step</th>
<th>Entered</th>
<th>Variable</th>
<th>Model R Square</th>
<th>F</th>
<th>p</th>
<th>Model R Square</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EXP</td>
<td>.16</td>
<td>7.48</td>
<td>.009*</td>
<td>.16</td>
<td>7.48</td>
<td>.009*</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CALC</td>
<td>.26</td>
<td>6.91</td>
<td>.003*</td>
<td>.10</td>
<td>5.50</td>
<td>.024*</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>WORK</td>
<td>.31</td>
<td>5.74</td>
<td>.002*</td>
<td>.05</td>
<td>2.77</td>
<td>.104</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>GRADE</td>
<td>.35</td>
<td>4.99</td>
<td>.003*</td>
<td>.04</td>
<td>2.19</td>
<td>.147</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A * B</td>
<td>.39</td>
<td>4.54</td>
<td>.003*</td>
<td>.07</td>
<td>3.88</td>
<td>.057</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Memory-B</td>
<td>.41</td>
<td>4.03</td>
<td>.004*</td>
<td>.02</td>
<td>1.27</td>
<td>.268</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CREDITS</td>
<td>.42</td>
<td>3.51</td>
<td>.006*</td>
<td>.01</td>
<td>0.67</td>
<td>.418</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>MemOps-A</td>
<td>.42</td>
<td>3.00</td>
<td>.012*</td>
<td>.00</td>
<td>0.05</td>
<td>.820</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ATT</td>
<td>.42</td>
<td>2.58</td>
<td>.023*</td>
<td>.00</td>
<td>0.00</td>
<td>.993</td>
<td></td>
</tr>
</tbody>
</table>

*p < .05.

Problem Solving Approaches

It was felt that a greater insight into the students' thinking could be obtained by examining how students attempted to solve the various exercises and programming problems which were done during the course of the study. Reported in this part of Chapter IV are the results of examining the methods which the students employed when solving the MemOps exercises. Also reported are the results of examining the protocols which the students used in writing the solutions to the Swap and the Simple Sort programming tasks.

MemOps Observations The first experimental treatment of this study was the use of the MemOps lesson. The purpose of this treatment was to encourage the student to discover how computer memory works and to gain an intuitive understanding of algorithm development. Using arrays of memory cells which first contained visible values and subsequently contained hidden values, the students were required to
place the values in ascending and descending order. During the hidden memory sorts, the students used a compare command to "ask" the computer which one of any two values stored in the array was larger. The compare command could be used as often as necessary in order to accomplish the sort.

All the steps which the students took to solve the MemOps exercises were automatically recorded by the computer in a history file. These history files were gleaned to determine how the students attempted to solve the sorting exercises. The methods which the students used to move the array values around were classified. These methods will be called fill methods from now on. In order to move the array values around, a temporary storage location had to be used. The use of temporary storage was also classified. How temporary storage was used will be referred to, hereafter, as storage methods.

A description of the fill and storage methods and how many students fell into the various classification groups are not provided. Instead, answers to the following questions were sought when the fill and storage methods were scrutinized. Did the students use the same fill method for both visible sorts? Did they use the same storage method for both visible sorts? Did they consistently use the same fill and storage methods for both hidden sorts?

There were 20 students who did the MemOps lesson. In comparing the fill methods for the visible memory sorts, only 4 students used the same fill method for both the ascending and descending sorts. Five students used the same storage methods. Only three students used the same fill method and the same storage method for both visible sorts.
In comparing the fill methods for the hidden memory sorts, the number of students who used the same fill method for both sorts was three. Eight students used the same storage methods. Only three students used the same fill method and the same storage method for both hidden sorts. Of the three students who were consistent for both the fill and storage methods for the visible sorts and of the three students who were consistent for both the fill and storage methods for the visible sorts, only one of these students was common to both groups.

The Swap  Students who obtained workable solutions to the Swap program used one of two methods to do the actual swapping of the two values. In the more commonly employed method, one temporary variable was used in conjunction with the two variables which contained the values to be interchanged. This method is called the Three Swap method. Students using the Three Swap method assigned the value of the first variable to the temporary variable, then the value of the second variable was assigned to the first variable, and finally the value of the temporary variable was assigned to the second variable. This sequence of three assignment statements resulted in swapping the values stored in the first and second variables. The Pascal code for the Three Swap is shown in Figure 23.

```
Temp := Num1;
Num1 := Num2;
Num2 := Temp;
```

Figure 23. Three Swap
In the other method, two temporary variables were employed. This method was called the Four Swap because four assignment statements and four variables were required. The two original values were first stored into two corresponding temporary variables. Then the first variable was assigned the value of the second temporary variable and the second variable was assigned the value of the first temporary variable. The Pascal code for the Four Swap is shown in Figure 24.

```
Temp1 := Num1;
Temp2 := Num2;
Num1 := Temp2;
Num2 := Temp1;
```

Figure 24. Four Swap

Of the 23 students who did a successful swap, 18 of them used the Three Swap method and the other five students used the Four Swap method. Of the five students who used the Four Swap, all five were Visible Memory students -- no Hidden Memory students did a Four Swap (Fisher's Exact p < .038). These data are shown in Tables 14 and 15.

Table 14. Summary of Obtaining Workable Solutions to the Swap Problem

<table>
<thead>
<tr>
<th>Obtained Solution</th>
<th>Visible Memory MemOps</th>
<th>Placebo</th>
<th>Hidden Memory MemOps</th>
<th>Placebo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>No</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Totals</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>12</td>
<td>9</td>
<td>11</td>
</tr>
</tbody>
</table>
Table 15. Summary of the Methods used to Solve the Swap Problem for the Students Who Obtained Workable Solutions

<table>
<thead>
<tr>
<th>Method Employed</th>
<th>Visible Memory</th>
<th>Hidden Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 swap</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>4 swap</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Totals</td>
<td>13</td>
<td>10</td>
</tr>
</tbody>
</table>

The types of errors which were made by students not obtaining a workable solution to the Swap were classified into six categories. The six categories are described below. A summary of the type of errors broken down by experimental treatment groups is shown in Table 16.

1. The first category contained the students who did not complete the program. Three students fell into this category; all three of them were in the Placebo group.

2. The students in the second category first assigned the two entered values to corresponding third and fourth variables. Then the variables were arranged in the WRITELN parameter list so that the fourth variable was placed before the third variable. Executing this program resulted in entering two values which were written out in reverse order. Eight students fell into the reverse output category.

3. The students in the third category first assigned the two entered values to corresponding third and fourth variables. Then the
third variable was placed prior to the fourth variable in the WRITELN parameter list. The result of running this program was such that the order of the values entered was the same as the order of the values which were written out. Two students fell into this category.

4. Students in the fourth category, called the Two Swap category, tried to do a swap without using a temporary variable. The incorrect solution consisted of assigning to the second variable the value of the first variable followed by assigning to the first variable the value of the second variable. The result was the loss of the value originally stored in the second variable. Five students committed this error; however, only one student was in the MemOps group. The Pascal code for a Two Swap is shown in Figure 25.

5. The fifth category contained only one student. The student in this category used English instead of Pascal to write the solution.

6. The sixth and last category contained one student who had a solution which was almost correct. This student did a correct Three Swap, but declared the temporary to be of type CHAR instead of type INTEGER. This student also used single quotes in the assignment statement.

The Simple Sort

Ten students obtained a workable solution to the Simple Sort problem. As reported in the findings in Part Two of this chapter, obtaining a workable solution to the Simple Sort was
Num2 := Num1;
Num1 := Num2;

Figure 25. Two Swap

Table 16. Summary of the Types of Errors Made by Students Who Did Not Obtain Workable Solutions to the Swap Problem

<table>
<thead>
<tr>
<th>Error Committed</th>
<th>Visible Memory</th>
<th></th>
<th>Hidden Memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MemOps</td>
<td>Placebo</td>
<td>MemOps</td>
<td>Placebo</td>
</tr>
<tr>
<td>Incomplete</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reversed output</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Unchanged order</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two swap</td>
<td>1</td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Used English</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Almost correct</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|       | 3 | 7 | Totals | 2 | 8 |

independent of student membership in either one of the two treatment groups. However, the algorithm used in the attempt to solve the Simple Sort was dependent on the use or non-use of MemOps.

There were essentially three different types of algorithms employed by the students when attempting to solve the Simple Sort. The first algorithm incorporated a swapping technique. The swapping technique consisted of checking the first two variables, if the first was larger than the second then their values were swapped. Next the
second variable was compared to the third variable. Again, if the second was larger than the third, then the two values were swapped. And finally, the first and second variables were again compared and if the first was larger than the second, then the values were interchanged. Only three students employed the swapping technique to sort the three values; two students were in the MemOps-Visible group and one student was in the Placebo-Hidden group.

The second algorithm consisted of several IF-THEN-ELSE statements -- whether in series, nested, or both in series and nested -- to determine the exact ordering of the three variables. Since there are six possible ways into which three values can be ordered, the IF-THEN-ELSE statements were organized in such a fashion that when the program ran, the flow of execution followed one of six unique and exclusive routes. At the end of a given route, the program "knew" the exact ordering of the three variables. What the students did at these six points in the program was to use WRITELN statements, one at each of the six points the program. Since the order of the three variables was known at each of the six points, the three variables were arranged in the WRITELN parameter list such that the three values were printed on the screen in ascending order. This algorithm is called the WRITELN algorithm.

The logic of the third algorithm, called the 1/2/3 algorithm, was parallel to logic used in the second algorithm described above. Several IF-THEN-ELSE statements were used to determine the exact ordering of the three variables. Since three separate values can be arranged in six possible ways, there were six different points where the program "knew"
the order of the three variables. Instead of using a WRITELN statement at each of the six different points in the program, the students used assignment statements. The purpose of the assignment statements was to store the ascending order of the three variables into a second set of variables. Then at the end of the program, a single WRITELN statement containing the second set of variables in the parameter list was used to print the three values in ascending order.

The one student who used an algorithm which was classified as "Other" used a combination of two of the above algorithms. The first step in this student's algorithm was to check if the first two variables were in ascending order and if they were not, a swap was done. Then, a series of three IF-THEN statements was used to determine where in relationship to the first two variables did the third variable belong: before, between, or after. Within each THEN part of the three IF-THEN statements was a WRITELN statement to print the three variables in ascending order.

The students who fell into one of the four categories discussed above employed algorithms which were easily discernible. The students who did not fall into one of these four categories attempted to solve the problem in a manner which could not be discerned from the programming protocols. Of the eight who fell into this unknown category, three were MemOps students and five were Placebo students.

The frequency counts of the algorithms used to solve the Simple Sort are shown in Table 17. Upon inspection of the frequency counts, it was observed that the cell counts for MemOps students using the 1/2/3 algorithm were larger than the cell counts for MemOps students using the
WRITELN algorithm. At the same time it was noticed that the WRITELN algorithm was used more often than was the 1/2/3 algorithm by the Placebo students. Analyzing these data with the Fisher's Exact procedure revealed that the probability of these two proportions being equal was .023. Therefore, the proportion of MemOps students using the 1/2/3 algorithm was significantly larger than the proportion of Placebo students who used the 1/2/3 algorithm. It was also noted that of the ten students who did obtain workable solutions to the Simple Sort, seven of those ten students used the 1/2/3 algorithm. Refer to Tables 17 and 18 for these data.

Table 17. Frequency Counts of the Algorithms Used to Solve the Simple Sort Programming Task by Experimental Group

<table>
<thead>
<tr>
<th>Approach Employed</th>
<th>Visible Memory</th>
<th>Hidden Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MemOps</td>
<td>Placebo</td>
</tr>
<tr>
<td>Swapping</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>WRITELN</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1/2/3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Other</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Unknown</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
Table 18. Summary of Obtaining Workable Solutions to the Simple Sort Problem

| Obtained Solution | Visible Memory | | Visible Memory | | | Hidden Memory | | | Hidden Memory |
|-------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|                   | MemOps         | Placebo        | MemOps         | Placebo        | MemOps         | Placebo        |
| Yes               | 1              | 4              | 3              | 2              |
| No                | 10             | 8              | 6              | 9              |
CHAPTER V. SUMMARY AND DISCUSSION

Summary

The problem of this study was to determine what effects the use of manipulative, dynamic computer-based models had on the meaningful learning of computer programming by novices. Two different computer-based models were used in this study: MemOps and MiniPas. The purpose of MemOps was to encourage the student to discover how computer memory works and to gain an intuitive understanding of algorithm development. The purpose of the second lesson, MiniPas, was to provide the novice programmer with a learning environment which was more transparent than the traditional programming environment.

The population for this study consisted of students enrolled in a first semester computer programming course. The sample was limited to those students enrolled in the Computer Applications in Industrial Education and Technology (IEDT 216) course at Iowa State University, Ames, Iowa. The subjects used for the study were 43 students, 42 males and one female, who took IEDT 216 during the spring semester of 1986.

The research design was a two by two factorial design. One main effect was the use or non-use of MemOps. The second main effect was the use of the visible or hidden memory version of MiniPas. Students were randomly assigned to one of the four experimental treatment groups using a stratified random assignment technique. Stratification was based on the level of computer-related experience which the student had prior to taking IEDT 216 and three categories were established. These categories
were (1) no experience, (2) some experience including BASIC programming, word processing, or data entry, and (3) Pascal or FORTRAN programming experience.

At the beginning of the semester, half the students received the MemOps experience and the other half received a placebo experience. After the MemOps/placebo experience, regular classroom instruction commenced. For the first half of the semester, the labs sections were split according to membership in the visible or hidden Minipas group. During this time, all students used MiniPas to complete their programming assignments. The split lab sections met concurrently at two separate lab sites; each section was taught by one of two instructors. An overt effort was made to minimize the differences between the concurrent classroom presentations by the use of identical lecture notes and overhead transparencies. To minimize the effect that the use of one lab site versus the other lab site may have had, the lab sections rotated lab sites once every two weeks. To minimize the effect of the instructors, the two instructors were rotated between the lab groups every week.

The dependent measures were obtained from (1) an inclass examination which consisted of short answer items and multiple choice items, (2) five novel programming tasks, and (3) programming protocols. The scores from the written examination were subjected to regression analysis using the two main effects and an interact effect as independent variables. The solutions to the novel programming tasks were scored dichotomously: either the student obtained a workable solution or the student did not obtain a workable solution. The
dichotomous scores were classified according to the student membership in the experimental treatment groups and then the data were analyzed using the Fisher's Exact procedure. The programming protocols employed by the students when solving the various programming tasks were gleaned to ascertain the type of programming practices used by beginning programming students.

Findings

The first research question was: Will there be a difference in learning between students using manipulative model and students who do not use the manipulative model? Based on the results of the regression analysis, it was concluded that there was no statistically significant difference attributable to the use or non-use of the manipulative model.

The second research question was: Will there be a difference in learning between students using visible memory and students who use the invisible memory? Based on the results of the regression analysis, it was concluded that there was no statistically significant difference attributable to the use of visible or hidden memory.

The third research question was: Will there be an interaction in learning between the students using or not using the manipulative model and students using visible or hidden memory? Based on the results of the regression analysis, it was concluded that there was no statistically significant interaction effect.

The conclusions of the first three research questions were similar; i.e., there were no significant differences in student learning
as measured by the inclass examinations scores which can be explained by
(1) the use or non-use of the manipulative model, (2) the use of hidden
or visible memory in MiniPas, and (3) the interaction of (1) and (2).
The conclusion that there were no differences in student learning which
were due to the two main effects and the interaction effect in no way
implies that no difference existed -- only those differences as measured
by the inclass examination did not exist.

The fourth research question was: Will there be a difference
between the proportion of MemOps students who obtain workable solutions
to novel programming tasks and the proportion of Placebo students who
obtain workable solutions? Based on the results of the Fisher's Exact
procedure, it was concluded that the proportion of MemOps students who
obtained a workable solution to the Swap programming problem was larger
than the proportion of Placebo student who obtained a solution.
However, the obtaining of workable solutions to the Simple Sort,
Compare, Reverse, and Array Sort programming tasks were independent of
using or not using MemOps.

The fifth research question was: Will there be a difference
between the proportion of visible memory students who obtain workable
solutions to novel programming task and the proportion of hidden memory
who obtain workable solutions? The conclusion of this research question
was that the use of visible and hidden memory in MiniPas was independent
of obtaining workable solutions to the Swap, Simple Sort, Reverse,
Compare, and Array Sort problems.

The sixth research question was: What type of programming
practices do beginning programming students use that may suggest
programming practices which are not typically used by experienced or expert programmers? Based on the examination of the programming protocols employed by the students, the following programming practices by beginning programming students were observed:

1. Several students used the FOR loop but failed to take advantage of the index of the FOR loop within the loop body.
2. Numerous students employed unique variable identifiers for indexes to FOR loops which were independent of each other.
3. Not all students understood that the control variable of one FOR loop was incremented independently of incrementing the control variable for a second FOR loop which was nested in the first FOR loop.
4. Two students attempted to implement one FOR loop with two control variables.
5. Several students initialized an integer variable to zero just prior to a FOR loop which used the integer variable as its index.
6. On numerous occasions, several students did not capitalize on the capabilities of the WRITELN statement to solve numeric expressions or to directly access array locations nor did students capitalize on the use of expressions as array subscripts.
7. When assigning the value of one variable to another variable, some students thought that some math operation had to be performed on the right side of the assignment operator.
8. Periodically, students used an assignment statement which assigned the value of a variable to itself.
9. Many students' interpretation of compiler error messages was literal only.

10. Some students never read the compiler error messages nor paid attention to where the syntax error was detected.

11. Many students made the same syntax error repeatedly throughout their programs, but failed to recognize in each instance that the same mistake was causing the compiler errors.

12. There was a considerable amount of variation in the manner in which the students test ran their programs.

Several post hoc analyses were conducted to explore additional effects that the manipulative models may have had on the student behaviors which were not revealed by the a priori analyses. The post hoc analyses included four different linear regression models and an examination of the problem solving approaches used by the students.

A summary of the findings of the post hoc regression models follows.

1. The first regression model, called the Full Model, contained nine independent variables: MemOps, Memory, the interaction between MemOps and Memory, CREDIT, WORK, EXP, CALC, ATT, and GRADE. The Full Model was significant and accounted for 42% of the variance in the examination scores. WORK, EXPER, and CALC were each significant after controlling for the other eight variables in the model.

2. In doing a stepwise forward regression analysis, the nine variables were entered into the successive models to maximize the
amount of variance accounted for in the criterion variable. The order in which the nine variables entered the stepwise model was EXP, CALC, WORK, GRADE, the interaction between MemOps and Memory, Memory, CREDIT, MemOps, and ATT.

A summary of the problem solving approaches used by the students when they worked on MemOps, the Swap and the Simple Sort are provided below.

1. Students were inconsistent in the methods which they choose to use when solving the MemOps sorting exercises.

2. Students who obtained workable solutions to the Swap problem used one or two temporary variables. All the students who used two temporary variables in their solutions were students who had used visible memory MiniPas. The types of errors made by the students who did not obtain workable solutions included (a) not using any temporary variables, (b) moving values around in memory but never interchanging the values, and (c) never completing the program.

3. As stated previously in the findings above, the use or non-use of MemOps was independent of obtaining a workable solution to the Simple Sort programming problem. However, the proportion of MemOps students who used additional memory locations in the attempt to solve the Simple Sort was significantly larger than the proportion of Placebo students who used additional memory locations.
Discussion

The discussion is organized into four parts according to the type of criterion variable examined.

Research Questions I, II, and III

Research Questions I, II, and III were used to investigate whether or not differences in learning existed which were due to the use or non-use of MemOps, the use of visible or hidden memory in MiniPas, or the interaction between these two main effects. The criterion variable was the inclass examination scores. The linear regression analysis revealed that there was no significant differences which were attributable to either one of the two main effects or which was attributable to the interaction effect.

All but one of the examination scores fell in the range of 14 points to the maximum of 24 points. The score which fell outside this range was 6 and the student receiving this score was in the Placebo-Visible group. By using the prediction equation and constructing a 95% confidence interval around the predicted score for a student in the Placebo-Visible group, the score of 6 fell outside the 95% confidence interval. Since the actual test for evaluating whether or not a given score was an outlier was more conservative than determining whether actual value fell into the 95% confidence interval for the predicted value, the score of 6 can be considered an outlier. However, a regression analysis with the score of 6 eliminated was not done for two reasons: (1) the small number of subjects in each of the
treatments, and (2) the variance in the criterion variable would be reduced considerably.

The variance for the MemOps group was 5.00 as compared to 17.30 for the Placebo group. However, after eliminating the score of 6, the variance of the Placebo group dropped to 8.49. The variance for the Visible group was 16.53 and after eliminating the score of six the variance dropped to 6.55.

The examination scores were all clustered at the high end of the scale. The students who received the maximum score may have experienced a "ceiling effect". If the variance of the examination scores were greater, the regression analysis may have been more sensitive to the treatment effects. The probability of obtaining the observed difference between the MemOps group and the Placebo group was 20% as compared to the 86% probability of obtaining the observed difference between the Memory groups. Therefore, the difference between MemOps and Placebo scores would have been more sensitive to an increased variance in the criterion variable than the differences between the Memory groups.

Research Questions IV and V

Research Question IV The fourth research question was: Will there be a difference between the proportion of MemOps students who obtain workable solutions to novel programming tasks and the proportion of Placebo students who obtain workable solutions? The only novel programming task which was statistically significant was the Swap. One of the exercises in MemOps was a swapping problem. Therefore, the transfer of learning from doing the MemOps swapping exercise to doing
the Pascal Swap program could explain why the proportion of MemOps students who obtained a workable solution to the Swap was larger than the proportion of Placebo students who obtained a workable solution. The Simple Sort, Reverse, and the Array Sort problems were chosen as criterion measures because the concept of a swap had to be applied in all three problems in order to obtain workable solutions. A significant proportion of students learned how to do the swap as evidenced by the results of the Swap problem, but the ability to apply the concept of the swap in a situation which differed from the context in which the concept was learned was lacking.

Research Question V The fifth research question was: Will there be a difference between the proportion of visible memory students who obtain workable solutions to novel programming tasks and the proportion of hidden memory students who obtain workable solutions? The frequencies for obtaining solutions for the visible and hidden groups were nearly identical. Therefore, there were no significant differences in the proportion of Visible Memory students who obtained workable solutions and the Hidden Memory students who obtained workable solutions. Merely providing the students with a more transparent learning environment obviously did not enhance the students' ability to obtain workable programming solutions. An overt effort was made throughout this experiment to minimize the instructional differences between the Visible Memory and the Hidden Memory groups. Had the instructors capitalized on the use of visible memory in MiniPas to help explain concepts pertaining to the storage of data in memory, the proportion of Visible Memory students who obtained workable solutions
may have been significantly different than the proportion of Hidden Memory students who obtained workable solutions. However, had visible memory in MiniPas been incorporated as part of the instruction, then the effects using or not using visible memory in MiniPas could not have been separated from the effects of integrating the instruction.

Since the use of the memory area in MiniPas was not formally accentuated by the instructors nor was it integrated with any of the students' learning activities, the students may have ignored the memory area. There are several possible reasons for having ignored it. One reason could be due to a mental overload. The students had to learn how to logon, how to use the operating system, how to move through CAS, how to use a new editor, how to use MiniPas, and so on. The students learned what was essential to get their daily assignments done. They found out that they could survive without learning how to use the flow of execution arrow and could survive without actively observing the memory area.

Some students may have attempted to use the memory area but became frustrated. If they couldn't figure out the connection between the program, the flow of execution arrow and the memory area, frustration occurred. Other explanation could be that the students just did not want to put forth the effort required to take advantage of the memory area. Since learning to program was already a monumental and time consuming task, why make it more so by paying attention to the special dynamics of MiniPas?
Research Question VI

The sixth research question was: What type of programming practices do beginning programming students use that may suggest programming practices which are not typically used by experienced or expert programmers? The importance of this research question was not merely to identify programming practices which beginning programming students used but rather the desire was to identify the practices in order to provide insight into the students' thinking about programming. The protocols have provided evidence to indicate what the students did. It was impossible to state in absolute terms what mental models the students were using to solve the problems. Therefore, only plausible hypotheses are described. The articulation of what beginning students have done will spawn future research studies to investigate the underlying causes for the observed student behaviors.

Several students used the FOR loop but failed to take advantage of the index of the FOR loop within the loop body. Instead of using the index, the students set up a second integer variable which was separate from the index but always had the same value as the FOR loop index. One hypotheses for this behavior was a lack of understanding of the FOR loop control structure. Students realized that the FOR loop can repeatedly do a segment of Pascal code a given number of times, but they did not realize that the value of the FOR loop index was accessible for use inside the body of the FOR loop. Another explanation for this behavior was simply a matter of the student breaking a more complex section of Pascal code into "mind size" pieces. The FOR loop control structure was
designed to count from a starting bound to an ending bound and the purpose of the index variable was to store the count values. From the perspective of a beginning programming student, the operation of the FOR loop was hidden from the student. It was hidden in the sense that the FOR loop automatically (1) initialized the index to the starting bound, (2) incremented the index every time at the end of the FOR loop body, and (3) decided whether or not to execute the body of the FOR loop again.

By setting up a second integer variable which was separate from the FOR loop index, the student could think of the FOR loop as controlling how many times the body of the FOR would be done. The second integer variable was doing a separate task, keeping track of the count. Since the student had to provide the assignment statements to initialize statements were more concrete to the student than what was automatically done by the FOR loop.

If the true motivation for using a second integer variable was to break the more complex FOR loop into smaller "mind size" pieces, then should the FOR loop be taught at the same time when the other looping control structures are taught? Should the students intentionally be taught how to use a WHILE loop instead of the FOR loop so that the student must explicitly use a counting variable and must provide all the assignment statements to manipulate the counting variable? Mayer (1979) has argued that more than just the statement and program levels of programming knowledge must be taught. According to Mayer, statements should be broken down into what he calls the machine, transaction, and prestatement levels and that statements should be put together into mandatory chunks, basic nonmandatory chunks, higher chunks, and
programs. If breaking the statements down into smaller pieces does improve student learning, the argument that machine language instruction should precede a higher level language would have more of a basis. In reporting the misconceptions which high school students who had had one semester of Pascal programming, Sleeman et al. (1986) stated that students thought the FOR loop control variable did not have a value inside of the loop body. Sleeman et al. did not provide additional details concerning the misconception nor did they venture an explanation.

Numerous students employed unique variable identifiers for indexes to FOR loops which were independent of each other. Whether or not these students realized that the control variables were also independent of each other cannot be ascertained from the protocols. The use of independent FOR loop control variables may be a programming aid which these students used to help themselves keep track of what each FOR loop was doing. More experienced programmers will use unique control variables to independent FOR loops in programs, especially large programs, as a means of internally documenting the program. For example, when a FOR loop is being used to address the rows in a matrix a control variable like "ROW" is used and when a FOR loop to address the columns is used a control variable like "COLUMN" is used. Another explanation for using unique control variables was as a safety factor. In the process of learning how to use FOR loops, either in lecture or from the textbook the student remembered that in certain situations unique control variables must be used. Instead of remembering what those special situations were, the student always used unique control
variables.

Not all students understand when the FOR loop control structure incremented the index of the FOR loop and that the incrementing of the index for two nested FOR loops was not concurrent. The use of BEGIN ENDS in Pascal to delimit the compound statements for the control structures may have been a contributing factor in making this error. For example, if two FOR loops are nested inside each other and the inside FOR loop body was a compound statement, the END serves as the END for both FOR loops. Another explanation for this error was that the student misunderstood the problem. The students may have thought that nested loops were needed to solve the problem.

Two students attempted to implement one FOR loop with two control variables. This error was made in the solution for the Compare problem. Since two separate arrays with unique names were being used, these students may have thought that unique array subscripts also had to be used. It was usual for students to construct a FOR loop in a form which was definitely different than what they had been doing for over ten weeks since they were first introduced to FOR loops.

Several students initialized an integer variable to zero just prior to a FOR loop which used the integer variable as its index. One explanation for this extra assignment statement was because the students did not realize that one of the things which FOR loops do is to initialize the control variable to the starting bound. Another explanation for using the assignment statement is as a safety factor. One of the rules beginning students encounter is to always initialize variables before the variables are used on the right hand side of other
assignment statements. Instead of trying to remember the conditions under which it was necessary to initialize variables to zero, students just did it all the time to be safe.

On numerous occasions, several students did not capitalize on the capabilities of the WRITELN statement to solve numeric expressions or to directly access array locations nor did they take advantage of using integer expressions as array subscripts. Instead, they assigned the value of the expression to a temporary variable and then used the temporary variable. Experienced programmers will typically use a temporary variable when an expression involves complicated calculations or when the expression is used repeatedly. This is done in the interest of speed and simplicity; plus, the internal documentation of the program is usually enhanced. One simple explanation for this behavior by novices is that they may not have realized that the syntax of these statements permitted expressions. (In matter of fact, compilers of yesteryear did not accommodate such syntax.)

Another explanation would be the breaking down of a more complex task into "mind size" pieces. It is easier for the students to do one task at a time in separate statements, rather than having one statement doing many things. Also, a series of simpler statements was easier to debug if the students did not know what they were doing, in this case, syntax. Another explanation deals with how the purpose of each Pascal statement was pictured in the student's mind. When the various Pascal statements were first presented in lecture, a general description of each statement's purpose was provided. For example, the purpose of WRITELN statements is to copy data which is stored in memory and write
it to the screen; whereas, the purpose of assignment statements is to move data around in memory or to do "number crunching." By placing an expression in a WRITELN statement, the WRITELN statement is doing something which the student remembers as being the purpose of an assignment statement.

Another hypothesis for this behavior was a desire by the student to store the results of expressions in an explicit and concrete memory location. The expert programmer knows that the various values in expressions are pushed onto and popped from the stack as the expression is evaluated. The final value of the expression is left on the stack to be retrieved when needed. The concept of a runtime stack which is maintained by the operating system is beyond the comprehension of a beginning programming student. In lecture, the students are told that they can use memory to store any values which they needed in their programs. This concept may have caused a conflict in their thinking since there were no definite memory locations for the values of expressions used in WRITELN statements or as array subscripts are stored.

Therefore, by first storing the results of an expression in a temporary variable and then using the temporary variable in the WRITELN or as an array subscript, the students had created a location in memory where they knew exactly where the needed value was stored. In other words, the students may have created a programming solution which fit their their mental models of how the computer functioned. The use of a temporary variable may have been especially meaningful to visible memory MiniPas students who were actively observing and studying the memory
area while using MiniPas. These students could see where the expression results were stored in memory by the presence of the memory box for the temporary variable.

When assigning the value of one variable to another variable, some students thought that some math operation had to be performed on the right side of the assignment operator. When the assignment statement was first learned, essentially all the examples which were given in the textbooks and in the lecture presentations were assignment statements with math operators on the right side of the assignment operator. There are few beginning level programming applications examples which require the copying of data stored in one memory location to another memory location. An assignment statement with only a single variable on the right side of the assignment operator is hereafter called a simple assignment statement. Two of the more common applications of simple assignment statements is to setup the proper conditions (1) before entering a WHILE loop and (2) at the end of the WHILE loop in preparation for the next iteration. In this course, as is the case in many beginning Pascal programming courses and textbooks, the topic of assignment statements and variables is covered immediately after WRITELN statements are first introduced. The topic of control structures, especially control structures which employ the use of simple assignment statements as discussed above, is not covered until later in the course. DuBois, Alverson, and Staley (1979) assert that concept learning is facilitated by the presentation of rational sets of instances and noninstances of the concept to be learned. If the above explanation is indeed true, then there is a definite need to provide
meaningful examples which require the use of simple assignment statements when they are first introduced. Another instructional ploy should also be considered. Maybe the original lecture presentation on assignment statements should exclude the use of simple assignment statements and delay presenting them until after there is a need created for the use of simple assignment statements.

Occasionally, students used an assignment statement which assigned the value of a variable to itself. One explanation is that the student thinks the value of a variable will run out or expire if it is not refreshed or used periodically. These students may not realize that once a value is stored at a particular memory location, that value will stay there for the duration of the program's execution unless the value is altered by an input or assignment statement. One student said that it never dawned on him that the values in memory could be used at any time whenever he needed them until he had used MiniPas for several weeks. He finally realized that the numbers were always over there [as he pointed to the MiniPas memory area on the screen] in the boxes whenever he wanted to used them.

Another skill which the novice programming students had to learn was the debugging of the programs which they wrote. A major part of the debugging process was interpreting and reacting to compiler error messages. Several students interpreted the error messages only in a literal fashion. If the compiler said Z was expected, the student added a Z, even though there already was a Z and adding another Z did not seem to make sense either. The first reaction to a compiler error message should be a literal reaction. However, knowing when to disregard the
literal meaning is a debugging skill which some students have great difficulty learning. The experienced programmer has learned how to think like the compiler "thinks;" therefore, the veteran programmer will not hesitate to ignore the literal meaning of the error message if it does not make sense in the given situation and interpret the error message from the compiler's perspective. Early in the semester, many students were literal interpreters, but as they gained more experienced they went beyond literal meanings of the error messages. However, some students never learned to go beyond the literal interpretations. One explanation for this is that these students were rule learners instead of rule makers. These students were merely cataloging syntax rules and not building a functional, integrated mental model which established the relationships among all the Pascal constructs being learned. They never were able to generalize the concepts and rules beyond the specific program that they were working on. The inability to generalize had a "snowball effect." When students fixed one part of their program, it can have an effect on a different part of the program. Without a mental model to tie all the details together, the students had to rely on hopefully finding a rule which would apply for the given situation. It is further hypothesized by this researcher that the students who never went beyond the literal interpretation of compiler error messages probably were the victims of educational systems which were built solely on the philosophy of behavioral psychology.

Some students never paid attention to the compiler error messages. The students who ignored the error messages typically were the students who struggled from the beginning of the course continued to
perform poorly. It was surmised that these students were never able to develop a mental model of computing that worked for them. Reasons for not developing a functional mental model may include (1) "computer anxiety," (2) missing one or two vital components in their mental models which were prerequisite to adding more intricate and advanced concepts and relationships to the model, or (3) simply a lack of time and effort on the students' behalf.

Another common mistake which was committed by the students was to make an error repeatedly throughout their entire program. Expert as well as novice programmers are guilty of this. The manner in which students corrected the repetitive error may be indicative of the students' level of understanding about programming. The students who encountered a syntax error and fixed all occurrences of the error throughout their program had a mental picture of the program. Whereas, the students who had to make several attempts (or guesses) in order to fix each occurrence of the identical error were not able to learn from first error occurrence. This latter group of students did not have a "hook" in their mental models of computing on which to "hang" what they were doing. This could also explain the difference between students who test run their programs with only one set of data versus the students who use numerous combinations of test data. If the program performed as the student expected with a given set of test data, he assumed the program was correct. However, the student who tested his program with several combinations of test data was able to visualize numerous relationships that existed among the data values which had to be tested in the program.
In summarizing the anecdote discussions, the following hypotheses of novice programming behaviors were noted.

1. Students had difficulty understanding statements which were "nested" and employed alternative algorithms to avoid nesting. A nested statement is defined as a statement which does more than one step or process. This definition is more comprehensive than traditional meaning of the nesting, e.g., nested FOR loops. This definition can refer to a single statement by itself. For example, a WRITELN which contains an expression which is evaluated before the resulting value is printed on the screen is considered a nested statement. Array subscripts which are expressions instead of a single value are also considered nested. A FOR loop in itself is a nested statement because of all the things which a FOR loop automatically does which are hidden from the programmer. The students tended to avoid the use of nested statements. They preferred to break these more complicated statements into multiple statements which were not nested.

2. Students were rotely applying arbitrary rules in their programs rather than creating generalized rules of their own.

3. Students apparently preferred to use temporary variables to store the values of expressions rather let the expressions be evaluated in WRITELN statements or allow expressions to be used as array subscripts. By storing the results of expressions in temporary variables, the students knew exactly where the value of the expression was located in memory.
Post Hoc Analyses

Several post hoc analyses were conducted in an effort to determine what effects the two computer lessons may have had on student behavior which were not revealed by the a priori analyses. The post hoc analyses consisted of constructing two additional regression models and examining the various problem solving approaches students employed when solving the sorting exercises in MemOps and when doing the Swap and Simple Sort programming tasks.

Regression Models

Two supplementary regression models were constructed by adding independent variables to the original regression model previously discussed. The purpose for including the additional independent variables was to increase the sensitivity of the analyses to the treatment effects.

The original regression model contained three independent variables: MemOps, Memory, and the interaction of MemOps and Memory. A regression model which contained six additional independent variables was constructed. This model was called the Full Model. The six variables were (1) the number of semester hours of credit taken during the spring semester of 1986 (CREDIT), (2) the total number of hours per week which the student had to work including volunteer activities, service as a club officer, or any other commitment which the student had on a regular weekly basis (WORK), (3) a dichotomous variable indicating whether or not the student had any prior computer-related experiences (EXP), (4) the number of semester hours of college calculus passed (CALC), (5) the score from an attitude scale used to determine the how
important the student felt computers would be in their college and professional careers (ATT), and (6) what grade the students expected to receive in IEDT 216 (GRADE).

The Full Model accounted for 42% of the variance in the examination scores. The Full Model accounted was statistically significant; however, the practical significance of this model was questionable for two reasons: first, a large number of independent variables relative to the total number of observations, and second, only 42% of the variance was accounted for in the examination scores. Cronbach and Snow (1977) considered aptitude-treatment interaction analyses which accounted for less than 50% of the variance to be weak models.

Three variables made significant contributions after controlling for the other eight variables in the model: EXP, CALC, and WORK. The amount of computer-related experience which the students had prior to taking this course should improve their performance on the examination; this result was not surprising. Neither was it surprising that the number of semester hours of college calculus passed made a positive contribution to doing well on the examination. The ability to manipulate variables and to abstractly think about these manipulations are skills which are common to both calculus and computer programming skills. Students who did not have a working knowledge of algebraic variables were at a disadvantage compared to the students who had a considerable amount of experience using them. The arguments which contend that the rigors of structured thinking and problem solving skills required to preform well in mathematics will facilitate similar
behavior when learning computer programming are not even remotely being suggested as a explanation for the effect of CALC. Such explanations are beyond the scope of this study.

The significant contribution of WORK after controlling for the effects of the other eight variables in light of the fact the CREDIT was not significant was unexpected. The regression coefficient for WORK was negative; in other words, the more hours which students worked per week at a parttime job, the poorer the students did on the inclass examination. These findings suggest that students sacrificed their time for studying in order to hold a parttime job. If the students did not show up for work, they would lose their jobs. However, if the students did not spend enough time studying, they were not dropped out of their classes.

The stepwise forward regression revealed EXP was the best one variable model which explained the largest amount of variance in the examination scores. This one variable model was significant. The best two variable model included the variables EXP and CALC. The additional amount of variance which the second variable CALC added to the one variable model was significant. The greater the amount of computer-related experiences which students had prior to taking IEDT 216 did enhance their performance on the examination. The ability to symbolically manipulate variables as done in algebra and calculus courses should, as expected, enhance the ability to understand and use computer variables. Entering the remaining variables into the stepwise models did not significantly increase the amount of variance explained in the examination scores. Therefore, the order in which the remaining
variables were entered was essentially meaningless -- especially in light of the large number of degrees of freedom in the model relative to the small sample size.

Problem Solving Approaches In an effort to gain a greater insight into the students' thinking, the methods which the students employed in their attempts to solve the sorting exercises in MemOps and the programming protocols from the Swap and the Simple Sort were examined.

In order to obtain correct solutions to the sorting exercises in MemOps, the students had to move the values around in the array. The methods which the students used to move the values was called fill methods. To prevent the loss of values stored in the array, temporary storage locations had to be used. The use of storage locations was called storage methods. The fill and storage methods which the students employed when they did the four sorting exercises were classified. Upon inspection of the fill and storage methods, it was discovered that only one student used the same fill and storage methods for all four sorting exercises. Only three students used the same fill and storage methods for both visible sorts and three students used the same fill and storage methods for both hidden sorts. In other words, the students in general were very inconsistent in the methods which they used to solve the sorting exercises. One explanation for the inconsistent behavior was that the students were trying different fill and storage methods in order to learn various ways that a sort could be done. However, this explanation is not too probable. If the students were intentionally testing various methods, then the particular set of values and the order
of the values in the array at the beginning of the exercise would not
dictate the method which the students would use to solve the problem.
For example, if the array values were already partially sorted or if
there were duplicate values in the array, certain short cuts could be
taken to obtain the final sorted order. When students realized that
these special situations existed, they capitalized upon them. The most
likely explanation for the students behavior is that they choose the
fill and storage methods which were the easiest for the situation at
hand.

Since the students were required to do each exercise only once,
there was insufficient evidence to state that student X has a tendency
to do sort Y using method Z. Instead, these results lead to more
unanswered questions. Why did the students choose a particular method
when doing the sort? Were the students thinking about how to generalize
the process of sorting or were they just trying to get "it done and over
with"?

A plausible hypothesis for a larger proportion of MemOps students
obtaining workable solutions to the Sort problem than Placebo students
is the near transfer of learning. One of the exercises which the MemOps
students had to do was a swap. Therefore, the obtaining of a solution
for MemOps students was merely a matter of translating the solution into
Pascal. The non-MemOps students had to figure out the algorithm and had
to translate the solution into Pascal. Of the 23 students who obtained
a workable solution to the Swap, only five of these students used the
Four Swap and all five students used visible memory in MiniPas. One
possible explanation for this is that students could breakdown the
problem into two parts, first, storing both values into temporary storage and then putting values back into the original variables. By having two temporary variables, students could see in the memory area a specific, temporary storage location for each of the two original values. The order in which the assignment statements are executed in the Four Swap is not as critical as the Three Swap order. There does not appear to be any rational explanation for this difference in terms of using visible versus hidden memory in MiniPas.

The explanations for making the errors which the students made in attempting to do the Swap are numerous and speculative. The students who never interchanged the values of the two variables but just stored the original two values into corresponding third and fourth variables probably never understood the problem in the first place. Either the problem was stated in a form which was confusing for these students or the students had such a weak understanding of how memory works that they had no concept of what it meant to swap the contents of two memory locations. The students who did a two swap obviously did not realize that storing a new value in a particular memory location destroyed the old value stored there. Another explanation for doing a Two Swap was that the students had an incorrect mental model of how computer memory operates. They may have thought that each memory location acted like a stack, i.e., the old value was still "under" the new value. This explanation was first articulated by Thomas, Boysen and Hooper (1985) as a result of interviewing students who initially pilot tested the MemOps lesson.

The proportion of MemOps students who obtained workable solutions
to the Simple Sort was nearly identical to the proportion of Placebo students who obtained workable solutions. Likewise, the proportion of visible and hidden Memory students obtaining workable solutions were nearly identical. However, there was a significant difference in the proportion of MemOps students who chose to use the WRITELN algorithm when attempting to solve the Simple Sort and the proportion of Placebo students who chose the WRITELN algorithm. MemOps students had a tendency to use the 1/2/3 algorithm instead of the WRITELN algorithm and Placebo students had the opposite tendency. These results suggest that the MemOps students had a better understanding of memory or, at least, saw the use of memory as one of the tools which could be utilized to solve the problem. Mayer's (1975) research indicated that students who received a concrete model prior to formal instruction could solve novel programming problems better than the students who received the concrete model after formal instruction. The fact that MemOps students had a stronger tendency to use a more complex means, 1/2/3 algorithm, parallels Mayer's findings.

Recommendations for Future Research

Very little is known about the cognitive aspects of how novices learn computer programming. The primary motivation for researching the programming process in the past has been to increase the efficiency of commercial software development teams. A considerable amount of research has examined the differences in the mental schemata of novices and experts from various content areas (Coombs, Gibson, and Alty, 1981). These research studies have provided some direction for research
studies such as this one, but future research studies need to focus on how the learning of computer programming actually affects the novices' mental schemata. To accomplish this objective, two types of research studies must be conducted: exploratory and inferential. Exploratory studies must be done to ascertain and articulate the behaviors of beginning programmers. Then based on the results of the exploratory studies, inferential experiments can be conducted to determine cause and effect.

The following recommendations are made on the basis of the findings and conclusions of this study and the experiences gained from conducting this experiment. The recommendations are divided into three sections. The first section contains recommendations for any future research which investigates how beginners learn to program -- regardless of whether it is exploratory or inferential research. The last two sections provide recommendations which are specific to each type of research study.

For Both Exploratory and Inferential Research

1. Future research studies need to concentrate on students who have not had any prior computer-related experiences. There are three reasons for using only novice subjects. First, understanding the novice learner is prerequisite to subsequent investigations which focus on how the more advanced programming skills are learned. The second reason is to reduce the amount of variance between subjects. The third reason is to study novices before they become extinct. As more and more students are exposed to computers at
home and school, finding students who have not had any computer-related experiences will become more difficult.

2. The educational and professional goals of the students participating in future research studies may influence the results. The role of the computer for a computer science student is different than for an IEDT student. Therefore, future research studies need to be sensitive to the perceptions which students have toward computing.

3. The validity and reliability of the instrument used to collect the student demographics needs to be examined.

4. The use of student demographics other than the ones used in this study merit investigation, e.g., student learning and problem-solving styles (sequential, random), student motivation and attitudes towards learning computer programming.

5. Future research studies must minimize the effect of extraneous factors which may effect the results. For example, the number of hours which the students worked each week at a part time job may obscure any experimental effects.

6. No attempt was made in this experiment to integrate the MemOps and the MiniPas experiences with the material which was formally presented in lecture. Future research must address effective means of integrating the use of computer-based lessons with other learning activities.

7. The effect of learning machine language prior to the learning of a high level language needs to be investigated. Some of the behaviors which the students exhibited in this research study
suggested that many students did not have a working knowledge of how computer memory operates. The purpose of MemOps and MiniPas was to provide an environment in which students could discover how memory functions. In both of these lessons, students were working with high level language programming constructs. Would working at a machine language level provide a window through which student could gain a more profound understanding of high level language programming? This researcher does not recommend that an actual machine language for a particular computer be learned. Rather, the recommendation is to use a computer-based lesson which contains a model of machine language programming. This is important for several reasons. The purpose for learning machine language would be to provide a foundation for future learning. Therefore, the emphasis would be on understanding the general concepts of machine language programming, not learning op codes for a particular central processing unit. The simulation can be designed to focus on the relevant concepts and the extraneous details which are inherent to regular machine language programming can be eliminated (e.g., base conversion, addressing modes, offset calculations, bit and byte manipulation). The machine language lesson must allow the student to directly manipulate and test the important concepts in an easy-to-use environment which is forgiving.

8. The essential nature of computer programming must be discovered. What are the fundamental underpinnings of computer programming? Without a profound understanding of the content area itself,
research efforts designed to improve the learning and the instruction of the content will be in vain.

For Exploratory Research

1. When collecting programming protocols for research purposes, forcing students to write "paper and pencil" solutions to programming problems and then allowing them to use MiniPas to obtain a final solution is recommended. Hooper (1986) used this technique and found it to be viable. Although several of the "paper and pencil" solutions were difficult to decipher, the written solutions did serve as the initial benchmark against which the final solutions could be compared. "Paper and pencil" solutions were not collected in this study and thus there were no initial solutions to use as benchmarks.

2. The use of student elaboration techniques may provide additional insights into the students' thinking. Student elaboration techniques could range from informal chats with the lab instructor, to formal interviews, to written summaries. The protocols provided a wealth of information, but they did not answer all the questions concerning the students' programming behaviors. By individually asking the students what they are doing and why they did particular things in their programs, may prove fruitful. However, there are several disadvantages of using student elaboration: (a) the elaboration itself may have an effect on subsequent learning, (b) novice students may not have the means to articulate their behaviors, and (c) the coordination
and implementation of elaboration techniques would not be trivial.

For Inferential Research

1. An experimental research study needs to be designed to investigate which skills, competencies, and experiences are prerequisite to the learning of computer programming.

2. An experimental research study needs to be designed which reveals in greater detail the mental models or schemata which beginning students employ for understanding how computer memory functions and how the flow of program execution works.

3. An experimental research study needs to be designed to investigate how beginning students learn to use the FOR loop. Some of the questions which need to be answered are: Why do students use a second integer variable instead of the FOR loop index? Why do students use unique control variables for independent FOR loops? Why do students initialize the control variable to zero just prior to the FOR loop in which the control variable is used? DuBoulay, O'Shea and Monk (1981) state that the notional machine must be revealed to the student by using a "glass box" approach rather than a "black box" approach. Since the FOR loop automatically maintains the control variable and thus these operations are hidden from the learner, could this be the "black box" approach to learning looping control structures? Whereas, should students first learn how to use a WHILE loop and manually maintain the control variable to emulate a FOR loop? Would this constitute a "glass box" approach?
4. As future research begins to reveal the nature of the novice programmer, what will be the implications for instructional methodologies? For example, one of the findings of the study was the use of temporary variables to store the values of expressions to be used in subsequent WRITELN statements or to be used as array subscripts. Should the textbook and lecture examples incorporate the use of temporary variables? If the reason the students are using temporary variables is so that all values are stored in an explicit memory location and thereby fitting their mental model of how memory works, then maybe the textbook and lecture examples should also use the temporary variables. On the other hand, it could be argued that the mental process that the student went through to generate the use of the temporary variable was the key to gaining a better understanding of programming. Future research needs to answer these questions.

Concluding Remarks

A tremendous amount of thought, planning, time, and energy went into the development of MemOps and MiniPas. The creating of a rich and powerful learning environment is a necessary first step, however, unless the student actively engages in the learning process, meaningful learning will not occur.

Mayer (1981) states that three conditions must be met in order for meaningful learning to occur. Learners must (1) receive or pay attention to the new information, (2) possess appropriate old knowledge or prerequisite concepts to which the new material can be anchored, and
(3) actively assimilate or connect the new knowledge to the old. Notice that the focus is on what the student must do, not on what the instructor must do. This does not imply, however, that the instructor's role is not important. The instructor's role is to facilitate the student's learning process. Regardless of what the instructor does and how well s/he does it, the student must do the learning.

One of the findings of this study was the use of temporary variables to store the results of expressions and then use the temporary variables in WRITELN statements or as array subscripts. If classroom teachers imply from these findings that they must adjust their instruction by incorporating the use of temporary variables, then this study will have been done in vain. Instead, the reaction should be "Why are the students using temporary variables?" Until future research discovers the mental models used by novice programmers, classroom teachers should permit students to use temporary variables. The effect of encouraging or discouraging the use of temporary variables on student learning is currently unknown. Therefore, the implications of this study for classroom teachers is to look beyond what the students are doing and ask why are the students learning in a particular way.
BIBLIOGRAPHY


Sherick, A. (1986). Classification Officer, Department of Industrial Education and Technology, Iowa State University, Ames, Iowa.


in computer conditionals—a psychological evaluation.  

Pascal and High School Students: A Study of Errors.  Journal of  
Educational Computing Research, 2(1), 5-24.

Soloway, E. (1985).  From Problems to Programs via Plans: The Content  
and Structure of Knowledge for Introductory LISP Programming.  

programming enhance problem solving ability? Some positive evidence  
on algebra word problems.  In Seidel, R. J., Anderson, R. E., and  
Hunter, B. (Eds.), Computer Literacy: Issues and Directions for  

University Press.

Sternberg, R. J. (1985b).  Teaching Critical Thinking, Part 2:  
Possible Solutions.  Phi Delta Kappan, 67(4), 277-280.

Computer-Supported Learning Group, Iowa State University Computation  
Center, Ames, Iowa.

Weinberg, G. (1971).  The psychology of computer programming.  New  
York: Van Nostrand Reinhold Company.

in novice and expert programmers.  International Journal of  
ACKNOWLEDGEMENTS

Many people contributed to the successful completion of this dissertation and I would like to acknowledge and thank these people. A sincere thank you goes to Dr. William Miller and Dr. Rex Thomas, my two major professors. Both these gentlemen spent many hours helping me these last few years and they have had a strong influence on my thinking and professional development during my graduate studies. I knew writing a dissertation was going to be a major undertaking, but nobody told me how much work it was going to be to coordinate two major professors. A very deserving thank you also goes to the other members of my committee: Dr. John Boysen, Dr. Ann Thompson, and Dr. Trevor Howe.

This experiment could not have been conducted without the students who participated in this study and the assistance of a second instructor, Dr. Elizabeth Hooper. I thank you, Lib -- not only for teaching one of the experimental groups but also for all the brainstorming sessions which help shape this research study.

Finally, my family deserves a warm thank you for their faith and support in me. Especially to my wife, Pam, for carrying the bulk of the parenting responsibilities while I worked on the degree. We had no children when I started my graduate studies and now we have three daughters: Sarah Jean, Erin Grace, and Abby Johanna. Now that the dissertation is complete, I am looking forward to being a full time, two parent family again.
APPENDIX A. HUMAN SUBJECTS APPROVAL

The Iowa State University Committee on the Use of Human Subjects in Research reviewed this project and concluded that the rights and welfare of the human subjects were adequately protected, that the risks were outweighed by the potential benefits and expected value of the knowledge sought, that confidentiality of data was assured and that informed consent was obtained by appropriate procedures.
One of the experimental treatments was the use of the visible and hidden memory versions of MiniPas. Since the students used MiniPas to complete their programming assignments for the first half of the semester, they had to be split into two lab sections. The students were randomly assigned to one of four color code groups and used the below lab schedule.

<table>
<thead>
<tr>
<th>Date</th>
<th>Group</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>WT Lab Jan 15/16</td>
<td>everybody</td>
<td>Quad E006</td>
</tr>
<tr>
<td>F Lec Jan 17</td>
<td>fill out survey</td>
<td>Quad E006</td>
</tr>
<tr>
<td>MT Lab Jan 20/21</td>
<td>everybody</td>
<td>Com Sci B29</td>
</tr>
<tr>
<td>WT Lab Jan 22/23</td>
<td>Gold &amp; Red</td>
<td>Gilman 357</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Quad E006</td>
</tr>
<tr>
<td>MT Lab Jan 27/28</td>
<td>Gold &amp; Red</td>
<td>Gilman 357</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Quad E006</td>
</tr>
<tr>
<td>WT Lab Jan 29/30</td>
<td>Gold &amp; Red</td>
<td>Quad E006</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Gilman 357</td>
</tr>
<tr>
<td>MT Lab Feb 3/4</td>
<td>Gold &amp; Red</td>
<td>Quad E006</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Gilman 357</td>
</tr>
<tr>
<td>WT Lab Feb 5/6</td>
<td>Gold &amp; Red</td>
<td>Quad E006</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Gilman 357</td>
</tr>
<tr>
<td>MT Lab Feb 10/11</td>
<td>everybody</td>
<td>Com Sci B29</td>
</tr>
<tr>
<td></td>
<td>Midterm #1</td>
<td>Gilman 357</td>
</tr>
<tr>
<td>WT Lab Feb 12/13</td>
<td>Gold &amp; Red</td>
<td>Quad E006</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Gilman 357</td>
</tr>
<tr>
<td>MT Lab Feb 17/18</td>
<td>Gold &amp; Red</td>
<td>Gilman 357</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Quad E006</td>
</tr>
<tr>
<td>WT Lab Feb 19/20</td>
<td>Gold &amp; Red</td>
<td>Gilman 357</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Quad E006</td>
</tr>
<tr>
<td>MT Lab Feb 24/25</td>
<td>Gold &amp; Red</td>
<td>Quad E006</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Quad E006</td>
</tr>
<tr>
<td>WT Lab Feb 26/27</td>
<td>Gold &amp; Red</td>
<td>Quad E006</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Quad E006</td>
</tr>
<tr>
<td>MT Lab Mar 3/4</td>
<td>Gold &amp; Red</td>
<td>Quad E006</td>
</tr>
<tr>
<td></td>
<td>Green &amp; Blue</td>
<td>Gilman 357</td>
</tr>
<tr>
<td>WT Lab Mar 5/6</td>
<td>everybody</td>
<td>Com Sci B29</td>
</tr>
<tr>
<td></td>
<td>Midterm #2</td>
<td>Quad E006</td>
</tr>
<tr>
<td>Mar 7-14</td>
<td>SPRING BREAK</td>
<td>Quad E006</td>
</tr>
</tbody>
</table>
APPENDIX C. COURSE OUTLINE

The topics which were covered in IEDT 216 for the duration of the experiment are listed below.

<table>
<thead>
<tr>
<th>WT Lab</th>
<th>Jan 15/16</th>
<th>logging on VAX, use of CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>F Lec</td>
<td>Jan 17</td>
<td>fill out survey, terminology: ROM, RAM, etc.</td>
</tr>
<tr>
<td>MT Lab</td>
<td>Jan 20/21</td>
<td>MemOps experimental treatment</td>
</tr>
<tr>
<td>WT Lab</td>
<td>Jan 22/23</td>
<td>general architecture of a computer, WRITELN</td>
</tr>
<tr>
<td>F Lec</td>
<td>Jan 24</td>
<td>parts of Pascal program, variable, math operators, assignment statements</td>
</tr>
<tr>
<td>MT Lab</td>
<td>Jan 27/28</td>
<td>IF THEN ELSE, boolean operators</td>
</tr>
<tr>
<td>WT Lab</td>
<td>Jan 29/30</td>
<td>IF THEN ELSE continued</td>
</tr>
<tr>
<td>F Lec</td>
<td>Jan 31</td>
<td>overview of control structures, REPEAT UNTIL</td>
</tr>
<tr>
<td>MT Lab</td>
<td>Feb 3/4</td>
<td>WHILE loop</td>
</tr>
<tr>
<td>WT Lab</td>
<td>Feb 5/6</td>
<td>FOR loop</td>
</tr>
<tr>
<td>F Lec</td>
<td>Feb 7</td>
<td>Review DCL commands</td>
</tr>
<tr>
<td>MT Lab</td>
<td>Feb 10/11</td>
<td>Swap experimental treatment</td>
</tr>
</tbody>
</table>
APPENDIX D. SURVEY FORM

A survey was used to collect demographic information from the students. The students were told that they would be using several computer-based lessons during the semester and in order to evaluate and improve the lessons, background information from the students was needed. Students filled out the survey during the first common lecture of the semester. A copy of the survey as given to the students follows on the next page.
This semester you will be using some new instructional computing materials to learn about computers and computer programming. These materials were developed to help alleviate some of the problems and misconceptions that previous students have encountered. As you work on the VAX computer, your programming solutions will automatically be collected for later analysis as part of a research project.

Each section of IEdT 216 will be split into two groups for many of the lab periods this semester. Lib Hooper will teach one lab section and Warner Smidt will teach the other section. These two instructors will alternate teaching the two sections.

The information requested on the attached survey questionnaire will be used to learn more about the background of the students enrolled in this course and will be used to help analyze the programs which you will be writing in this course. This information and any other data that are collected from you will be kept strictly confidential.

Thank you for your cooperation, Warner Smidt.

Directions: Please complete the following items to the best of your knowledge.

NOTE: All information which you provide on this research questionnaire will be kept in strict confidence and will have absolutely no bearing in determining your course grade.

1. Signature_________________________ 2. Age_______

3. Sex: M or F  4. Classification: Fresh Soph Jr Sr Grad


7. How many credits of coursework are you taking this semester? _____ semester hours

8. How many hours per week do you work? (Also include volunteer work, service as a representative on Government of Student Body, or any other commitments which you have on a regular basis.) _____ hours per week

9. Did you have a microcomputer available for your use in your home while you were in high school? _____ yes _____ no

10. Do you currently have microcomputer available for your use where you live while going to ISU? _____ yes _____ no
11. What high school computer science courses have you taken? 
(Please describe the major activities of each.)

12. What college computer science courses have you taken? 
(Please describe the major activities of each.)

13. What other experience have you had with computers? (List any course-related or job-related activities such as use of wordprocessor for writing papers, use of a data base or spreadsheet programs, use of a statistical package for a statistics course, etc.)

14. If you have computer programming experience, please check all languages in which you have written programs.

   ____ BASIC   ____ Pascal   ____ FORTRAN   ____ COBAL
   ____ PL/1    ____ C       ____ machine language ____ LOGO
   ____ Others, specify: ______________________________________

15. Place a check beside all of the mathematics courses you took (and successfully passed) in grades 9-12.

   ____ Algebra I   ____ Algebra II   ____ Geometry  ____ Calculus
   ____ General    ____ Business     ____ Trigonometry
    Mathematics    Mathematics

   ____ Others, specify: ______________________________________
16. Please list all of the mathematics course which you have taken in college. Please include the number of semester hours for each course.

17. Place a check beside your current college GPA.

- 3.5 to 4.0
- 3.0 to 3.45
- 2.5 to 2.99
- 2.0 to 2.49
- 1.5 to 1.99
- Below 1.5

18. Circle the grade that you expect to receive in this course.

A  A-  B+  B  B-  C+  C  C-  D+  D  D-  F

19. Why are you taking this course? Check all that apply.

- to meet general computer literacy requirement
- to meet computer programming requirement
- to meet a 200 level course requirement
- to meet a free elective
- prerequisite requirement for other course(s)
- I know how to program but need to learn Pascal
- Other, specify: __________________________

20. Check the statement which describes how you plan to use computers in the future. Check all that apply.

- I don't plan to use or need a computer again.
- I will need to use a computer while in college, but will not need it once I leave ISU.
- I will need to use a computer again, but someone else will operate the computer.
- I will need to use a computer again and I may or may not be the operator.
- I will not use computers directly; but I will need to have a working knowledge of computers to succeed in a future job.
- My job will center around the use and applications of computers.
- I will be programming computers in my future job.
- Other, specify: __________________________
Directions: Circle a number between 1 to 9, inclusive, which best represents your feelings or attitudes.

21. I feel this course will improve my academic performance while in college.

<table>
<thead>
<tr>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly agree</td>
<td>neutral</td>
<td>disagree</td>
<td>strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

22. I feel this course will improve my chances of getting the job that I want after graduation.

<table>
<thead>
<tr>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly agree</td>
<td>neutral</td>
<td>disagree</td>
<td>strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

23. I feel this course will improve my opportunities for job advancements and promotions.

<table>
<thead>
<tr>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly agree</td>
<td>neutral</td>
<td>disagree</td>
<td>strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

24. I do not plan to be a professional programmer; therefore, I will not need an extensive working knowledge of computers and their applications.

<table>
<thead>
<tr>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly agree</td>
<td>neutral</td>
<td>disagree</td>
<td>strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

25. Other people will do all the computer work for me (secretaries, computer scientists or engineers); therefore, I will not need an extensive working knowledge of computers and their applications.

<table>
<thead>
<tr>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly agree</td>
<td>neutral</td>
<td>disagree</td>
<td>strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX E. SWAP TEST

The swap test was the first of the two novel programming tasks which the students did as part of the midterm examination. The swap problem as posed to the students is provided below. The students used MiniPas to solve the problem. MiniPas maintained a history file containing the programming protocols as each student worked. The history files were examined to determine (1) if each student obtained a workable solution and (2) what programming practices the students employed.

Program Problem #1 (Time Limit: 30 minutes)

Write a program which inputs from the user two numbers, swaps the two numbers around, and outputs the two numbers. The following analogy explains what is meant by the term swap. If you and I are each wearing a cap and we decide to swap caps, the cap which you had is now mine and my old cap is now yours.

Let's call the first number inputted by the user Num1 and the second Num2. Assume, for example, that the user inputs the number 5 into the variable Num1 and the number 10 into the variable Num2. After your Pascal program does the swap, the variable Num1 will contain the number 10 and the variable Num2 will contain the number 5.

Whether you use reals or integers is not critical; writing the Pascal to do the swap is the important part of this problem.
APPENDIX F. SIMPLE SORT TEST

The simple sort test was the second of the two novel programming tasks which the students did as part of the midterm examination. The simple sort problem as posed to the students is provided below. The students used MiniPas to solve the problem. MiniPas maintained a history file containing the programming protocols as each student worked. The history files were examined to determine (1) if each student obtained a workable solution and (2) what programming practices the students employed.

Program Problem #2 (Time Limit: 60 minutes)

Write a program which inputs any three numbers from the user and then outputs these three numbers in sorted order. The smallest number will be outputted first, the middle number outputted second, and finally the largest number will be outputted last.

For example, if the user enters the numbers 8 5 6, the output will be 5 6 8. If the input is 6 4 2, the output will be 2 4 6; if the input is 7 9 7, the output will be 7 7 9. Whether you use reals or integers is not important. Being able to write the Pascal to sort any three numbers into ascending order, however, is important.
APPENDIX G. COMPARE TEST

The compare test was the first of the three novel programming tasks which the students did as part of the final examination. The compare problem as posed to the students is provided below. The students did not use MiniPas for this problem. The solutions were hand written only.

**Problem #1** - Write Pascal code that will compare the contents of X[1] to Y[1], X[2] to Y[2], etc. and print out a message after each comparison stating which one contains the larger of the two integers stored in each array. Assume the following declarations have been made.

```pascal
CONST Max = 7;
VAR X, Y : ARRAY[1..Max] OF INTEGER;
    I, J, K : INTEGER;
```
APPENDIX H. REVERSE TEST

The reverse test was the second of the three novel programming tasks which the students did as part of the final examination. The reverse problem as posed to the students is provided below. The students did not use MiniPas for this problem. The solutions were hand written only.

Problem #2 - Part of a program that will REVERSE the order of the values stored in array X appears below. (If the values in X were 2, 4, 9, 10, 16 then the program segment below would reverse them so that X would now contain 16, 10, 9, 4, 2.) Fill in the bounds to the FOR statement that would be required to perform this reversal and add whatever Pascal statements are necessary to complete the reversal. You may use ONLY those constants, arrays, and variables that have been declared in the code below. You MAY NOT declare any additional ones. Use your own input data for this problem.

```
PROGRAM ReverseM (INPUT, OUTPUT);
CONST Max = 10;
VAR X : ARRAY[1..Max] OF INTEGER;
I, J, N, R : INTEGER;
BEGIN
FOR I := 1 TO MAX DO READLN(X[I]);
FOR I := __________ TO __________ DO
BEGIN
  (* Add the statements needed to complete REVERSEM below this line. *)
END;
END;
```
APPENDIX I. ARRAY SORT TEST

The array sort test was the third of the three novel programming tasks which the students did as part of the final examination. The compare problem as posed to the students is provided below. The students used MiniPas for this problem.

Problem #3 - Write a Pascal program that will put the values stored in an integer array of size 6 in order such that the smallest values stored in the array is located in the first element of the array and the largest value stored in the array is located in the last element of the array. You cannot use any constants, arrays, or variables other than those that have been declared for you in the code below. Use your own input data for this problem.

PROGRAM PutInOrder(INPUT,OUTPUT);
CONST Max = 6;
VAR X : ARRAY[1..6] OF INTEGER;
   I,J,Z,R : INTEGER;
BEGIN
   FOR I := 1 TO Max DO READLN(X[I]);
APPENDIX J. EXAMPLE OF A MEMOPS HISTORY FILE

The use or non-use of MemOps was the experimental treatments. While students worked through the exercises, MemOps maintained a history file containing all the commands which the students issued. An example history file is provided below.

USER GROUP: XYZ STUDENT'S NAME: XYZ32
22-JAN-1986 14:39:14

 visible memory operations GOAL: FIND MINIMUM TRY#1 14:44:48
GIVEN ARRAY: 9 46 45 5 68 TARGET ARRAY: -- -- -- --
MOVED X[4] TO Z USER ARRAY: 9 46 45 5 68 5
RIGHT >>> TARGET ARRAY: -- -- -- -- -- USER ARRAY: 9 46 45 5 68 5

visible memory operations GOAL: FIND MAXIMUM TRY#1 14:46:01
GIVEN ARRAY: 51 94 75 96 72 TARGET ARRAY: -- -- -- -- --
MOVED X[4] TO Z USER ARRAY: 51 94 75 96 72 96
RIGHT >>> TARGET ARRAY: -- -- -- -- -- USER ARRAY: 51 94 75 96 72 96

visible memory operations GOAL: SWAP TRY#1 14:46:58
GIVEN ARRAY: 5 33 0 0 0 TARGET ARRAY: 33 5 -- -- --
MOVED X[1] TO X[2] USER ARRAY: 5 5 0 0 0 --
WRONG >>> TARGET ARRAY: 33 5 -- -- -- USER ARRAY: 5 5 0 0 0 --

visible memory operations GOAL: SWAP TRY#2 14:50:30
GIVEN ARRAY: 5 33 0 0 0 TARGET ARRAY: 33 5 -- -- --

visible memory operations GOAL: SWAP TRY#3 14:51:20
GIVEN ARRAY: 5 33 0 0 0 TARGET ARRAY: 33 5 -- -- --

visible memory operations GOAL: SORT ASCENDING TRY#1 14:52:52
GIVEN ARRAY: 58 90 76 89 34 TARGET ARRAY: 34 58 76 89 90 --

visible memory operations GOAL: SORT DESCENDING TRY#1 14:53:27
GIVEN ARRAY: 78 41 20 16 50 TARGET ARRAY: 78 50 41 20 16 --

hidden memory operations GOAL: FIND MINIMUM TRY#1 14:56:24
GIVEN ARRAY: 35 97 66 31 27 TARGET ARRAY: -- -- -- --
MOVED X[5] TO Z USER ARRAY: 35 97 66 31 27 27
RIGHT >>> TARGET ARRAY: -- -- -- -- -- USER ARRAY: 35 97 66 31 27 27

@@@@ HIDDEN MEMORY OPERATIONS GOAL: FIND MAXIMUM TRY#1 14:58:32
GIVEN ARRAY: 11 53 79 75 90 TARGET ARRAY: -- -- -- -- --
MOVED X[3] TO Z USER ARRAY: 11 53 79 75 90 79
RIGHT >>> TARGET ARRAY: -- -- -- -- -- USER ARRAY: 11 53 79 75 90 79

@@@@ HIDDEN MEMORY OPERATIONS GOAL: FIND MAXIMUM TRY#2 15:01:12
GIVEN ARRAY: 11 53 79 75 90 TARGET ARRAY: -- -- -- -- --
MOVED X[5] TO Z USER ARRAY: 11 53 79 75 90 90
RIGHT >>> TARGET ARRAY: -- -- -- -- -- USER ARRAY: 11 53 79 75 90 90

@@@@ HIDDEN MEMORY OPERATIONS GOAL: SORT ASCENDING TRY#1 15:03:44
GIVEN ARRAY: 44 41 79 27 92 TARGET ARRAY: 27 41 44 79 92
MOVED X[1] TO Z USER ARRAY: 44 41 79 27 92 44
MOVED Z TO X[3] USER ARRAY: 27 41 44 79 92 44
RIGHT >>> TARGET ARRAY: 27 41 44 79 92 44

@@@@ HIDDEN MEMORY OPERATIONS GOAL: SORT DESCENDING TRY#1 15:12:00
GIVEN ARRAY: 49 85 47 73 49 TARGET ARRAY: 85 73 49 49 47
SUM_ONE 1 15:13:19
SUM_ONE 0 15:13:46
SUM_ONE 2 15:14:10
SUM_ONE 2 15:14:28
22-JAN-1986 15:15:05
APPENDIX K. EXAMPLE OF A MINIPAS HISTORY FILE

Students used the MiniPas programming environment to complete their programming assignments during the first half of the semester. MiniPas was also used for solving three of the novel programming tasks which were part of the midterm and final examinations. MiniPas maintained a history file containing copies of programs which did not compile due to syntax errors and copies of programs which were executed along with all data entered by the students when they executed their programs. An example history file is provided below.

XYZ36 entered MiniPas on 7-MAY-1986 at 11:06:08.71
************************************************************************** Compile: ';' expected. Line: 6 Column: 0 11:20:34.25 PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
   I,J,Z,R : INTEGER;
   Switiches : BOOLEAN;

BEGIN

FOR I:= 1 TO Max DO READLN(X[I]);

REPEAT
   J:= 2;
   FOR I:=1 TO (Max-1) DO
      BEGIN
         Switiches := FALSE;
         IF X[I] > X[J] THEN DO
            BEGIN
               Z := X[I];
               X[I] := X[J];
               X[J] := Z;
               Switiches := TRUE;
            END;
      END;

END;
J:= J+1;
END;
UNTIL Switches := FALSE
END.
*******************************************************************************
Compile: SWITCHES is undefined. Line: 16 Column: 12 11:21:32.69
PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
    I,J,Z,R : INTEGER;
    Switches : BOOLEAN;
BEGIN
FOR I:= 1 TO Max DO READLN(X[I]);
REPEAT
    J:= 2;
    FOR I:=1 TO (Max-1) DO
        BEGIN
            Switches := FALSE;
            IF X[I] > X[J] THEN DO
                BEGIN
                    Z := X[I];
                    X[I] := X[J];
                    X[J] := Z;
                    Switches := TRUE;
                END;
            J:= J-1;
        END;
    UNTIL Switches := FALSE;
END.
*******************************************************************************
PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
    I,J,Z,R : INTEGER;
    Switches : BOOLEAN;
BEGIN
FOR I:= 1 TO Max DO READLN(X[I]);
REPEAT
    J:= 2;

FOR I:=1 TO (Max-1) DO
BEGIN
Switches := FALSE;

IF X[I] > X[J] THEN DO
BEGIN
  Z := X[I];
  X[I] := X[J];
  X[J] := Z;
  Switches := TRUE;
END;
J:= J+1;
END;
UNTIL Switches := FALSE;
END.

******************************************************************************


PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
    I,J,Z,R : INTEGER;
    Switches : BOOLEAN;
BEGIN
FOR I:= 1 TO Max DO READLN(X[I]);
Switches := TRUE;

WHILE Switches DO
Begin
  J:= 2;
  FOR I:=1 TO (Max-1) DO
  BEGIN
    Switches := FALSE;

    IF X[I] > X[J] THEN DO
    BEGIN
      Z := X[I];
      X[I] := X[J];
      X[J] := Z;
      Switches := TRUE;
    END;

    J:= J+1;
  END;
END;
END.
PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
    I,J,Z,R : INTEGER;
    Switches : BOOLEAN;
BEGIN
    FOR I:= 1 TO Max DO READLN(X[I]);
    Switches := TRUE;
    WHILE Switches DO
    BEGIN
        J:= 2;
        FOR I:=1 TO (Max-1) DO
        BEGIN
            Switches := FALSE;
            IF X[I] > X[J] THEN DO
            BEGIN
                Z := X[I]
                X[I] := X[J]
                X[J] := Z
                Switches := TRUE
            END;
            J:= J+1;
        END;
    END;
END.

Compile: 'END' expected. Line: 20 Column: 0 11:29:05.83
PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
    I,J,Z,R : INTEGER;
    Switches : BOOLEAN;
BEGIN
    FOR I:= 1 TO Max DO READLN(X[I]);
    Switches := TRUE;
    WHILE Switches DO
    BEGIN
        J:= 2;
        FOR I:=1 TO (Max-1) DO
        BEGIN
            Switches := FALSE;
            IF X[I] > X[J] THEN DO
            BEGIN
                Z := X[I]
                X[I] := X[J]
                X[J] := Z
                Switches := TRUE
            END;
            J:= J+1;
        END;
    END;
END.

Compile: 'END' expected. Line: 23 Column: 0 11:35:22.89
PROGRAM PutInOrder (Input, Output);
BEGIN
  J := 2;

  FOR I := 1 TO (Max - 1) DO
  BEGIN
    Switches := FALSE;
    IF X[I] > X[J] THEN DO
      BEGIN
        Z := X[I]
        X[I] := X[J]
        X[J] := Z
        Switches := TRUE
      END;
    J := J + 1;
  END;
END;
END.

***********

PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
  I, J, Z, R : INTEGER;
  Switches : BOOLEAN;
BEGIN
  FOR I := 1 TO Max DO READLN(X[I]);
  Switches := TRUE;
  WHILE Switches DO
  BEGIN
    J := 2;
    FOR I := 1 TO (Max - 1) DO
    BEGIN
      Switches := FALSE;
      IF X[I] > X[J] THEN 
      BEGIN
        Z := X[I]
        X[I] := X[J]
        X[J] := Z
        Switches := TRUE
      END;
    J := J + 1;
  END;
END;
END.
PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
    I,J,Z,R : INTEGER;
    Switches : BOOLEAN;
BEGIN
    FOR I := 1 TO Max DO READLN(X[I]);
    Switches := TRUE;
    WHILE Switches DO BEGIN
        J := 2;
        FOR I := 1 TO (Max-1) DO BEGIN
            Switches := FALSE;
            IF X[I] > X[J] THEN BEGIN
                Z := X[I];
                X[I] := X[J];
                X[J] := Z;
                Switches := TRUE;
            END;
            J := J+1;
        END;
    END;
END.

Student Input: 34
Student Input: 4
Student Input: 6
Student Input: 23
Student Input: 5
Student Input: 17
Successful Compilation 11:45:49.23
Successful Compilation 11:47:32.62

RUN:11:47:35.53
PROGRAM PutInOrder (Input, Output);

CONST Max = 6;
VAR X: ARRAY [1..6] OF INTEGER;
    I,J,Z,R : INTEGER;
    Switches : BOOLEAN;
BEGIN

FOR I:= 1 TO Max DO READLN(X[I]);
Switches := TRUE;

WHILE Switches DO
BEGIN
  J:= 2;
  FOR I:=l TO (Max-1) DO
  BEGIN
    IF X[I] > X[J] THEN
    BEGIN
      Z := X[I];
      X[I] := X[J];
      X[J] := Z;
      Switches := TRUE;
    END;
    J:= J+1;
    Switches := FALSE;
  END;
END;

END.

Student Input: 12
Student Input: 2
Student Input: 5
Student Input: 8
Student Input: 3
Student Input: 17

XYZ36 exited MiniPasc on 7-MAY-1986 at 11:49:19.23
APPENDIX L. ALGORITHMS AND DATA STRUCTURES EMPLOYED
IN THE CLOSURE PROGRAM

Using arrays of memory cells which contained hidden values, the students were required to place the values in ascending and descending order. The students used a compare command to "ask" the computer which of one of any two values in the array was larger. The compare command could be used as often as necessary in order to accomplish the sort. MemOps maintained a history of all the commands which the students issued while working through the exercises. In order to determine whether or not the students had made enough "compare" commands to so that they absolutely knew the order of the values in the array, the Closure Program was designed and implemented. The students are said to have attained closure when they had made enough "compare" commands to know the absolute ordering of the values stored in the array.

The basic algorithm for the Closure Program was taken from Data Structures and Algorithms by Aho, Hopcroft, and Ullman (1983). The algorithm is founded in mathematical graph theory. Definitions for graph, tree, nodes, edges, etc. have not been provided; instead, you are referred to the above reference for these definitions.

The purpose of the basic algorithm, called the Warshall algorithm, is to determine if there exists a path between all pairs of nodes in a directed graph. The data structure which is used by the Warshall algorithm to represent the directed graph is an incidence matrix. Using the Closure Program essentially consists of performing four steps:
1. entering the required data;
2. submitting the incidence matrix which contains all the "compare" commands to the Warshall algorithm;
3. computing whether or not the matrix returned from the Warshall algorithm contains "closure paths"; and
4. reporting to the user the results of the analysis.

The comprehensive details concerning the data structure and the algorithm which were employed follow.

Three groups of data were required as input to the Closure Program: (1) the array values which the student had to sort into order, (2) all the "move" and "compare" commands which the student issued for a given exercise, and (3) whether an ascending or descending sort was being done. This data was stored as a directed graph. Each value in the array represented a node in the graph. Each comparison (made by the student in MemOps) between two values in the array represented an directed edge in the graph. For ascending sorts, the direction of the edge was from a node containing the smaller element to a node containing the larger element. For descending sorts, the direction of the edge was in the opposite direction: from the node containing the larger element to a node containing the smaller element.

The directed graph was implemented by using a Boolean matrix, also called the incidence matrix. The dimensions of this matrix were five rows by five columns -- one row and one column for each value stored in the MemOps X array. The rows represented the tail (or origin) of each directed edge in the graph. The columns represent the head (or
destination) of each directed edge in the graph. At the beginning of
the Closure Program, all 25 values in the Boolean matrix were
initialized to false indicating that no comparisons have been made. See
Figure L-1.

Figure L-1. The Initialized State of the Incidence Matrix

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

False indicates that no edge is present. This matrix contains
all falses; therefore, the graph which it represents contains zero
edges.

Each comparison which is made by the student was represented by a
ture value in the appropriate row and column. For example, assume a
student is sorting the X array into ascending order. The student
X[5]. To represent this comparison in the matrix, a value of true would
be assigned to the second row, fifth column. The second row marks the
tail of the edge going from the node containing the smaller value to the
column which represents the head of the edge pointing to the node
containing the larger value. A graph representing these two nodes and
the comparison is shown in Figure L-2. The storage of this same graph
in the data structure is shown in Figure L-3.

Figure L-2. A Graph Representing One Directed Edge
True in row 2, column 5 represents the tail of a directed edge originating from node 2 and the head terminating at node 5.

Figure L-3. An Incidence Matrix Representing One Directed Edge

After all the comparisons which were made by the student were stored in the incidence matrix, the matrix was then processed by the Warshall algorithm. The easiest way to explain what the Warshall algorithm did to the incidence matrix is to explain it by example. Assume that a student made two comparisons: X[2] with X[5] and X[5] with X[3]. A graph representing these two comparisons is shown in Figure L-4. From Figure L-4, it can be observed that a path exists from node 2 to node 3 by going through node 5. These two comparisons are stored as directed edges which are represented as two true values in the incidence matrix. This matrix is shown in Figure L-5.

Figure L-4. A Graph Representing Two Directed Edges
Two Edges represented in this graph:

1. from node 2 to node 5

2. from node 5 to node 3

Figure L-5. An Incidence Matrix Representing Two Directed Edges

The purpose of the Warshall algorithm is to find paths which connect all pairs of nodes by indirectly passing through other nodes; in this case, the path from X[2] to X[3] via X[5] should be found. The original matrix which contains the two edges, i.e., the two comparisons made by the student, is submitted to the Warshall algorithm. Afterwards, the resulting matrix contains the original two edges plus a third edge, from X[2] to X[3]. The third edge was added to the original matrix since there exists a path from X[2] to X[3] by passing through X[5]. In general, the resulting matrix will always contain at least all the edges which were in the original matrix and if path(s) exist which connect any two pairs of nodes together by passing through other node(s), then these edges are also added to the matrix by the Warshall algorithm. You are referred to Data Structures and Algorithms by Aho, Hopcroft, and Ullman for an explanation of the logic and a proof of the Warshall algorithm. Figure L-6 contains the resultant matrix.
After the incidence matrix is processed by the Warshall algorithm, the incidence matrix must be examined to determine whether or not the student attained closure when doing the respective MemOps sorting exercise. The logic of computing closure is a special application of graph theory which was developed for use in this research study by this investigator. In terms of graph theory, two conditions must be met before closure can be attained. These two conditions are based on the resultant matrix, not the original incidence matrix. The first condition is that paths must exist from any one node to all the remaining nodes. The second condition is if the direction of all the edges (in the resultant matrix) are reversed, paths must exist from any one node to all the remaining nodes. Computing to determine if these two conditions exist is relatively simple. If there is one row and one column in the resultant matrix which contains four true values, then closure was attained.

An example printout of the results of the Closure Program is provided in Figure L-7. A listing of the program is provided in Appendix M.
Hidden Sort Ascend Try#1  

Student: ABC12

Comparisons Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO --&gt;</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>FROM</td>
<td>26</td>
<td>42</td>
<td>44</td>
<td>15</td>
<td>50</td>
</tr>
</tbody>
</table>

8 edges total

Path Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO --&gt;</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>FROM</td>
<td>26</td>
<td>42</td>
<td>44</td>
<td>15</td>
<td>50</td>
</tr>
</tbody>
</table>

10 edges total

Figure L-7. Example Printout from the Closure Program
APPENDIX M. CLOSURE PROGRAM LISTING

The Closure Program was a tool which was used when the MemOps history files were analyzed. A listing of this program is provided below. An explanation of the algorithms and data structures used in the Closure program is provided in Appendix L. The UTILITY.PAS file which was included by the compiler directive, (*$I UTILITY.PAS*), is provided in Appendix R.

(*$u-,r+*)
PROGRAM Logic1(INPUT,OUTPUT);

CONST
  MaxValues = 6;
  MaxRecords = 100;
  MaxSteps = 23;

TYPE
  ExerciseType = ARRAY[1..9] OF STRING[20];

  AdjacentType = RECORD
    M : ARRAY[1..MaxValues, 1..MaxValues] OF BOOLEAN;
    Size : INTEGER;
    Name : Str80;
  END;

  PairType = RECORD
    Action : CHAR;
    First, Second : 0..6; (* X[1..5] and Z *)
  END;

  PairArrayType = ARRAY[1..MaxSteps] OF PairType;

  ProbType = RECORD
    Values : ARRAY[1..MaxValues] OF INTEGER;
    VSize, PSize, Exercise, Try : INTEGER;
    Pair : PairArrayType;
    Name : STRING[10];
  END;

VAR
  HaveClosure, Empty, Modified,
  HaveNumbers, HaveName, Loaded, Ok : BOOLEAN;
Choic : CHAR;
OldSize, PrintCount, LoadRecNum : INTEGER;
ListFileName : Str80;
Exercise : ExerciseType;
List : ProbType;
Data, Closure : AdjacentType;
ListFile : FILE OF ProbType;

PROCEDURE PageCheck(Need : INTEGER);
BEGIN
IF PrintCount * 15 + Need > 60
THEN
BEGIN
WRITELN(Device, CHR(12));
PrintCount := 0;
END;
END;

PROCEDURE InputRange(VAR Int : INTEGER;
Lo, Hi : INTEGER);
VAR
Status : INTEGER;
TempStr : Str80;
BEGIN
REPEAT
READLN(TempStr);
Val(TempStr, Int, Status);
WHILE Status <> 0 Do
BEGIN
WRITELN('Invalid number given, try again');
READLN(TempStr);
Val(TempStr, Int, Status);
END;
IF NOT(Int IN [Lo..Hi])
THEN WRITELN('Number must be in the ', Lo, '..', Hi, ' range');
UNTIL Int IN [Lo..Hi];
END;

PROCEDURE Warshall(C : AdjacentType;
VAR A : AdjacentType);
VAR
I, J, K : INTEGER;
Name : STRING[20];
BEGIN
Name := A.Name;
A := C;
A.Name := Name;
FOR K := 1 To A.Size DO
FOR I := 1 TO A.Size DO
    FOR J := 1 TO A.Size DO
        IF NOT A.M[I,J] THEN
            BEGIN
            END;
    END;

PROCEDURE MakeDataMat(L : ProbType; VAR A : AdjacentType);
VAR
    Continue : BOOLEAN;
    R,C,First,Second : INTEGER;
BEGIN
    FOR R := 1 TO MaxValues DO
        FOR C := 1 TO MaxValues DO
            A.M[R,C] := FALSE; (* assume no adjacency *)
    A.Size := 5;
    I := 0;
    Continue := I < L.PSize;
    IF Continue THEN Continue := Continue AND (L.Pair[I].Action = 'O);
    WHILE Continue DO
        BEGIN
            I := I + 1;
            First := L.Pair[I].First;
            Second := L.Pair[I].Second;
            IF L.Exercise IN [2,7,4,8] (* visible find max & hidden find max
            (* visible ascending sort & hidden ascending sort *)
            THEN
                ELSE
                    ELSE
                        BEGIN
                            A.M[First,Second] := TRUE;
                            A.M[Second,First] := TRUE
                        END
                END
            ELSE
                ELSE
                    ELSE
                        BEGIN
                            A.M[First,Second] := TRUE;
                        END
        END
A.M[Second,First] := TRUE;
END;
Continue := I < L.PSize;
IF Continue
THEN Continue := Continue AND (L.Pair[I+1].Action = 'C');
END;
END;

PROCEDURE ShowMat(D : ProbType;
A : AdjacentType;
ShowPaths : BOOLEAN;
VAR HaveClosure : BOOLEAN);
VAR
SolidRow, SolidCol : BOOLEAN;
R,C,Nodes,TotalNodes : INTEGER;
Row,Col : ARRAY[1..MaxValues] OF BOOLEAN;
BEGIN
PageCheck(15);
WRITELN(Device);
Pad(A.Name, 18,' ', 'R', A.Name);
WRITE(Device, A.Name);
WRITE(Device, Exercise[D.Exercise]:22, ' Try#', D.Try);
WRITE(Device, ' Student: ', D.Name, ' RN#');
IF Loaded
THEN WRITELN(Device, LoadRecNum+1)
ELSE WRITELN(Device, '?');
TotalNodes := 0;
FOR R := 1 TO 5 DO
BEGIN
Nodes := 0;
FOR C := 1 TO 5 DO
IF A.M[R,C] AND (R<>C)
THEN Nodes := Nodes + 1;
Row[R] := (Nodes = 4);
TotalNodes := TotalNodes + Nodes;
END;
IF ShowPaths
THEN
BEGIN
FOR C := 1 TO 5 DO
BEGIN
Nodes := 0;
FOR R := 1 TO 5 DO
IF A.M[R,C] AND (R<>C)
THEN Nodes := Nodes + 1;
Col[C] := Nodes = 4;
END;
SolidRow := FALSE;
FOR R := 1 TO 5 DO SolidRow := SolidRow OR Row[R];
END;
SolidCol := FALSE;
FOR C := 1 TO 5 DO SolidCol := SolidCol OR Col[C];
END;
IF A.Size = 0 THEN WRITELN(Device, 'empty') ELSE
BEGIN
WRITELN(Device);
WRITE(Device, '§ TO -->§ ');
FOR C := 1 TO A.Size DO
WRITE(Device, C:3);
WRITELN(Device);
WRITE(Device, '§ +§ FROM § ');
FOR C := 1 TO A.Size DO
WRITE(Device, D.Values[C]:3);
HaveClosure := ShowPaths AND SolidCol AND SolidRow;
IF HaveClosure THEN WRITELN(Device, ' Student has closure')
ELSE WRITELN(Device);
WRITE(Device, '-----++--');
IF ShowPaths THEN
FOR C := 1 TO A.Size DO
IF Col[C] THEN WRITE(Device,'--*')
ELSE WRITE(Device,'---')
ELSE WRITE(Device,'----------');
WRITELN(Device,'- TotalNodes, ' edges total');
FOR R := 1 TO A.Size DO
BEGIN
WRITE(Device, R:3, ' §', D.Values[R]:3);
IF ShowPaths THEN
IF Row[R] THEN WRITE(Device,' * ') ELSE WRITE(Device,' § ')
ELSE WRITE(Device,' § ');
FOR C := 1 TO A.Size DO
IF R = C THEN WRITE(Device,'':3)
ELSE IF A.M[R,C] THEN WRITE(Device,'1':3)
ELSE WRITE(Device,'0':3);
WRITELN(Device); END;
END; (* if a.size <> 0 *)
IF ShowPaths THEN
BEGIN
WRITELN(Device, '*' - '---
WRITELN(Device);
END;
PrintCount := PrintCount + 1;
PageCheck(0);
END;

FUNCTION GetNumber(Lo, Hi : INTEGER) : INTEGER;
VAR
  N : CHAR;
BEGIN
  REPEAT
    READ(KBD,N);
  UNTIL (ORD(N) - ORD('0')) IN [Lo..Hi];
  WRITE(N);
  GetNumber := CRD(N) - ORD('0');
END;

PROCEDURE Init;
VAR
  R, C : INTEGER;
BEGIN
  Exercise[1] := 'Visible Minimum';
  Exercise[8] := 'hidden Sort Ascend';
  Exercise[9] := 'Hidden Sort Descend';
  SstUpper;
  ListFileName := 'B:Memops1.dat';
  Loaded := FALSE;
  Data.Size := 0;
  FOR R := 1 TO MaxValues DO
    FOR C := 1 TO MaxValues DO
      Data.M[R,C] := FALSE;
  Closure := Data;
  Data.Name := 'Comparisons Matrix';
  Closure.Name := 'Path Matrix';
  List.PSize := 0;
  List.Name := 'empty';
  List.Exercise := 0;
  List.Try := 0;
  HaveName := FALSE;
  HaveNumbers := FALSE;
Modified := FALSE;
Empty := TRUE;
END;

PROCEDURE ReInit; (* for start of new student, so goto top of
printer form and send all output to CRT *)
BEGIN
CLOSE(Device);
Assign(Device,'LST:');
REWRITE(Device);
IF Printer AND (PrintCount > 0)
THEN WRITELN(CHR(12));
CLOSE(Device);
Assign(Device, 'CON:');
REWRITE(Device);
Printer := FALSE;
PrintCount := 0;
END;

PROCEDURE Create;
VAR
   Ok : BOOLEAN;
BEGIN
WRITELN;
Ok := TRUE;
WRITELN('Place data disk in drive B:');
PPReturnC;
IF FilePresent(ListFileName)
THEN
   BEGIN
      WRITE('"', ListFileName, '" already exists, ok to delete?');
      Ok := YesNo = Yes;
      IF Ok
      THEN
         BEGIN
            WRITE('VERIFY: ok to delete?');
            Ok := YesNo = Yes;
         END;
      END;
   IF Ok
   THEN
      BEGIN
         Assign(ListFile, ListFileName);
         REWRITE(ListFile);
         CLOSE(ListFile);
         WRITELN('File created');
      END;
   END;
END;
PROCEDURE InputComparisons(VAR L : ProbType);
     (* don't need VAR, but is faster *)
CONST
   Last = 6;
VAR
   Line, Option, I : INTEGER;
   Item : PairType;
  (* local *) PROCEDURE WriteIt(P : PairType; I : INTEGER);
BEGIN
   WBlanks(70,1+1,9);
   GOTOXY(70,1+1);
   WITH P DO
     WRITE(Action, First:3, Second:3);
END;
  (* local *) PROCEDURE Verify;
VAR I : INTEGER;
BEGIN
   WBlanks(40,1,34);
   GOTOXY(40,1);WRITE(L.Name:10, ' ', Exercise[L.Exercise]:20,
                   ' Try#', L.Try, L.PSize:3);
   FOR I := 1 TO L.PSize DO
     WriteIt(L.Pair[I], I);
END;
  (* local *) PROCEDURE Ask(VAR Item : PairType; Quit: BOOLEAN);
VAR
   Line : INTEGER;
   Choices : SetOfChar;
BEGIN
   Choices := ['m','M','C','c'];
   IF Quit
     THEN Choices := Choices + ['Q','q'];
   Line := 15;
   GOTOXY(1,Line);
   WRITE('action (M/C): ');
   Item.Action := Upper(GsTchar(Ghoicss)) ;
   IF Item.Action <> 'Q'
     THEN
       BEGIN
         GOTOXY(l,Line+1);WRITE(' first: ');
         Item.First := GetNumber(l,6);
         GOTOXY(l,Line+2);Write(' second: ');
         Item.Second := GetNumber(l,6);
       END;
   END;
  (* local *) PROCEDURE Insert;
VAR X,I, QuitLine : INTEGER;
BEGIN
   GOTOXY(4,22);writeln('Where one to insert');
QuitLine := L.PSize+2;
GOTOXY(70,QuitLine);WRITE('Quit');
X := Index(70,2,L.PSize+1,5024,L.PSize DIV 2);
IF X IN [1..L.PSize] THEN
BEGIN
  Ask(Item,FALSE);
  FOR I := L.PSize DOWNTO X DO
    L.Pair[I+1] := L.Pair[I];
  L.Pair[X] := Item;
  L.PSize := L.PSize + 1;
  Verify;
  Modified := TRUE;
END;
GOTOXY(66,X+1);WRITE(' '); (* erase pointer arrow *)
WBlanks(4,22,30);
END;
(* local *) PROCEDURE Add;
Begin
  GOTOXY(1,13);WRITELN('C(ompare, M(ove, Q(uit');
  Ask(Item,TRUE);
  WHILE Item.Action='Q' AND (L.PSize < MaxSteps) DO
    BEGIN
      L.PSize := L.PSize + 1;
      L.Pair[L.PSize] := Item;
      WriteIt(Item,L.PSize);
      WBlanks(78,1,3);
      GOTOXY(78,1);WRITE(L.PSize:3);
      Ask(Item,TRUE);
      Modified := TRUE;
    END;
  WBlanks(1,13,40);
END;
(* local *) PROCEDURE Delete;
VAR X,I, QuitLine : INTEGER;
BEGIN
  GOTOXY(4,22);WRITELN('Which one to delete?');
  QuitLine := L.PSize+2;
  GOTOXY(70,QuitLine);WRITE('Quit');
  X := Index(70,2,L.PSize+1,5024,L.PSize DIV 2);
  IF X IN [1..L.PSize] THEN
    BEGIN
      WBlanks(1,22,30);
      GOTOXY(1,22);WRITE('Ok to delete');
      IF YesNo = Yes THEN
        BEGIN
          WBlanks(70, L.PSize+1,10); (* erase last item in list *)
          L.PSize := L.PSize - 1;
        END;
FOR I := X TO L.PSize DO
    L.Pair[I] := L.Pair[I+1];
    Verify;
    Modified := TRUE;
END;
END;
WBlanks(1,22,30);
GOTOXY(70,QuitLine);WRITE(''); (* erase 'Quit" *)
GOTOXY(66,X+1);WRITE(''); (* erase pointer arrow *)
END;

(* local *)
PROCEDURE Replace;
VAR X,I, QuitLine : INTEGER;
BEGIN
GOTOXY(4,22);WRITELN('Which one to replace?');
QuitLine := L.PSize+2;
GOTOXY(70,QuitLine);WRITE('Quit');
X := Index(70,2,L.PSize+1,5024,L.PSize DIV 2);
IF X IN [1..L.PSize] THEN
    BEGIN
        GOTOXY(1,13);WRITELNC('C(ompare or M(ove');
        Ask(Item,FALSE);
        L.Pair[X] := Item;
        Writetxt(Item,X);
        Modified := TRUE;
    END;
END;
WBlanks(4,22,25);
WBlanks(1,13,25);
GOTOXY(70,QuitLine);WRITE(''); (* erase 'Quit" *)
GOTOXY(66,X+1);WRITE(''); (* erase pointer arrow *)
END;

BEGIN
ClrSCR;
WRITELN('Actions Editor');
Line := 3;
GOTOXY(5,Line+ 0);WRITELN('add');
GOTOXY(5,Line+ 1);WRITELN('insert');
GOTOXY(5,Line+ 2);WRITELN('delete');
GOTOXY(5,Line+ 3);WRITELN('replace');
GOTOXY(5,Line+ 4);WRITELN('verify');
GOTOXY(5,Line+ 5);WRITELN('Quit this menu');
GOTOXY(32,1);WRITELN('X Values');
FOR I := 1 TO 5 DO
    BEGIN
        GOTOXY(32,I+1);WRITELN(I, ' ', List.Values[I]:3)
    END;
Verify;
Option := 1;
REPEAT
    Option := Index(4,3, Last, 24, Option);
    END;
}
CASE Option OF
  1 : Add;
  2 : Insert;
  3 : Delete;
  4 : Replace;
  5 : Verify;
END;
UNTIL Option = Last;
Empty := FALSE;
END;

PROCEDURE NewName;
VAR
  I : INTEGER;
BEGIN
  WRITELN;
  WRITELN('Old username is ', List.Name);
  WRITE('Enter new username: ');
  READLN(List.Name);
  WRITELN('Old exercise is ', List.Exercise);
  FOR I := 1 TO 9 DO
    WRITELN(I:2, ' ', Exercise[I]);
  WRITE('Enter new exercise: ');
  List.Exercise := GetNumber(1,9);
  WRITELN;
  WRITE('Try#: ');
  List.Try := GetNumber(1,9);
  WRITELN;
  Modified := TRUE;
  HaveName := TRUE;
END;

PROCEDURE NewNumbers;
VAR
  R : INTEGER;
BEGIN
  WRITELN;
  WRITELN('Enter values in X array');
  FOR R := 1 TO 5 DO
    BEGIN
      WRITE(R:2, ' : ');
      InputRange(List.Values[R], 1,99);
    END;
  List.VSize := 5;
  HaveNumbers := TRUE;
  Modified := TRUE;
END;
PROCEDURE Append;
VAR
  RecNum : INTEGER;
BEGIN
  WRITELN;
  WRITE('Ok to append');
  IF YesNo = Yes THEN
    BEGIN
      Assign(ListFile, ListFileName);
      RESET(ListFile);
      WRITELN('Placed into record#', FileSize(ListFile)+1);
      LoadRecNum := FileSize(ListFile);
      Loaded := TRUE;
      SeekCListFile, LoadRecNum);
      WRITE(ListFile, List);
      CLOSE(ListFile);
      Modified := FALSE;
    END;
  END;
END;

PROCEDURE Save;
VAR
  RecNum : INTEGER;
BEGIN
  WRITELN;
  WRITE('Ok to save to record#', LoadRecNum+1);
  IF YesNo = Yes THEN
    BEGIN
      Assign(ListFile, ListFileName);
      RESET(ListFile);
      Seek(ListFile, LoadRecNum);
      WRITE(ListFile, List);
      CLOSE(ListFile);
      Modified := FALSE;
    END;
  END;
END;

PROCEDURE Load;
VAR
  FSize,RecNum : INTEGER;
BEGIN
  Assign(ListFile, ListFileName);
  RESET(ListFile)
  FSize := FileSize(ListFile);
  WRITELN;
WRITE('Load which record number? (1.., FSize,')
InputRange(RecNum, 1, FSize);
Seek(ListFile, RecNum-1);
READ(ListFile, List);
CLOSE(ListFile);
Loaded := TRUE;
LoadRecNum := RecNum-1;
Modified := FALSE;
HaveNumbers := TRUE;
HaveName := TRUE;
ReInit;
END;

PROCEDURE Analysis;
Var
  HaveClosure, Found : BOOLEAN;
  OldSize : INTEGER;
BEGIN
IF HaveNumbers THEN
  Begin
    Writeln('Doing Analysis...');
    OldSize := List.PSize;
    List.PSize := 0;
    Found := FALSE;
    WHILE NOT Found AND (List.PSize < OldSize) DO
      BEGIN
        List.PSize := List.PSize + 1;
        Found := List.Pair[List.PSize].Action = 'M';
      END;
    List.PSize := List.PSize - 1;
    HaveClosure := TRUE;
    WHILE HaveClosure DO
      BEGIN
        MakeDataMat(List, Data);
        Warshall(Data,Closure);
        ShowMat(List,Data,FALSE,HaveClosure);
        ShowMat(List,Closure,TRUE,HaveClosure);
        List.PSize := List.PSize - 1;
      END;
    List.PSize := OldSize;
  END
ELSE WRITELN('Must have numbers first');
END;

PROCEDURE Examine(L : ProbType);
VAR
  ZUsed : BOOLEAN;
V, A, Top, One, Two : INTEGER;
Old : PairArrayType;
BEGIN
WRITELN('Doing Examination...');
PageCheck(4 + L.PSize * 2);
WRITELN(Device);
WRITE(Device, 'MemOps Action ', Exercise[L.Exercise], ' Try#', L.Try,
    ' Student: ', L.Name, ' RN#');
IF Loaded
THEN WRITELN(Device, LoadRecNum+1)
ELSE WRITELN(Device, '?');
WRITELN(Device);
FOR V := 1 TO 5 DO
    WRITE(Device, L.Values[V];3);
WRITELN(Device); WRITELN(Device);*)
Top := 5;
ZUsed := FALSE;
FOR A := 1 TO L.PSize DO
    WITH L.Pair[A] DO
BEGIN
    WRITE(Device, A;4,'. ', Action, First:3, Second:3, ' ');
    IF Action = 'C'
    THEN
        BEGIN
            One := L.Values[First];
            Two := L.Values[Second];
            WRITE(Device, One;3);
            IF One < Two
            THEN WRITE(Device, ' <')
            ELSE
                IF One > Two
                THEN Write(Device, ' >')
                ELSE Write(Device, ' =');
            WRITE(Device, ', ', Two: 2, ';5);
            FOR V := 1 TO Top DO
                BEGIN
                    WRITE(Device, L.Values[V];4);
                END;
            END;
        END;
    ELSE
        BEGIN
            L.Values[Second] := L.Values[First];
            IF Second = 6
            THEN
                BEGIN
                    L.Values[First] := L.Values[Second];
                    IF First = 6
                    THEN
                        BEGIN
                            L.Values[Second] := L.Values[First];
                            IF Second = 6
                            THEN Write(Device, ' ')
                            ELSE Write(Device, ' ');
                        END;
                    END;
                END;
            ELSE
                BEGIN
                    L.Values[Second] := L.Values[First];
                    IF Second = 6
                    THEN Write(Device, ' ')
                    ELSE Write(Device, ' ');
                END;
            END;
        END;
    END;
END
THEN Top := 6;
WRITE(Device, '".';'13);
FOR V := 1 TO Top DO
BEGIN
WRITE(Device, L.Values[V], 4);
IF V = Second
THEN WRITE(Device, '*')
ELSE IF V = First
THEN Write(Device, ' .')
ELSE WRITE(Device, ' ');
END;
WRITELN(Device);
END; (* for *)
WRITELN(Device);
END; (* with *)
WRITELN(Device);
IF (L.PSize > 15) AND Printer
THEN
BEGIN
WRITELN(Device, CHR(12));
PrintCount := 0;
END
ELSE PrintCount := PrintCount + 2;
END;
BEGIN (* main body *)
InitUtility;
Init;
PrintCount := 0;
Assign(Device, 'CON:',)
REWITE(Device);
Printer := FALSE;
WRITELN;
WRITELN('Sort Analyzer');
WRITELN; WRITELN('<Ctri-A> to create a new data file');
WRITELN;
REPEAT
WRITELN;
WRITE('RN$');
IF Loaded
THEN WRITE(LoadRecNum+1:3)
ELSE WRITE('?':3);
WRITE(' C(hange username, S(ave, L(oad, N(ew, P(rint, ',
' eXamine, D(evice');
WRITE(' G(et numbers, E(dit actions, F(igure, A(nalyze, Q uit: ');
Choic := Upper(GetChar([CHR('I, 'C', 'S', 'L', 'P', 'N', 'G',
' E', 'D', 'F', 'A', 'Q'])));
WRITELN; WRITELN;
IF Choic = CHR(1)
THEN Create;
CASE Choic OF
   'X' : Examine(List);
   'N' : BEGIN
      Ok := TRUE;
      IF Modified
      THEN
         BEGIN
            WRITE('Ok to abandon changes to ', List.Name);
            Ok := YesNo = Yes;
         END;
      IF Ok
      THEN
         BEGIN
            Init;
            ReInit;
            WRITELN('Ready for new exercise...');
         END;
      END;
   'C' : NewName;
   'G' : BEGIN
      IF NOT HaveName
      THEN
         NewName;
         NewNumbers;
      END;
   'E' : BEGIN
      IF NOT HaveName
      THEN NewName;
      IF NOT HaveNumbers
      THEN NewNumbers;
      InputComparisons(List);
      CLRSCR;
      END;
   'F' : BEGIN
      IF HaveNumbers
      THEN
         BEGIN
            WRITELN('Doing Warshall Algorithm...');
            MakeDataMat(List, Data);
            Warshall(Data,Closure);
            ShowMat(List,Data,TRUE,HaveClosure);
            ShowMat(List,Closure,TRUE,HaveClosure);
         END
      ELSE WRITELN('Must have numbers first');
      END;
   'A' : Analysis;
   'D' : BEGIN
      WRITE('Send output to P(rinter or S(creeen? ');
}
IF GetChar(['P','S','C','M']) IN ['S','C','M']
THEN
BEGIN
WRITELN;
WRITELN('Output device is the screen');
CLOSE(Device);
Assign(Device, 'CON:');
REWRITE(Device);
Printer := FALSE;
END
ELSE
BEGIN
WRITELN;
WRITELN('Output device is the printer');
CLOSE(Device);
Assign(Device, 'LST:');
REWRITE(Device);
Printer := TRUE;
IF NOT Loaded AND NOT Empty
THEN Append;
END;
ENDIF
'P' : BEGIN
IF Printer THEN WRITELN('Printing...');
ShowMat(List, Data, FALSE, HaveClosure);
ShowMat(List, Closure, TRUE, HaveClosure);
END;
'S' : BEGIN
IF FilePresent(ListFileName)
THEN
BEGIN
IF Loaded
THEN
BEGIN
WRITE('S(ave as record#', LoadRecNum+1,' or A(ppend?');
IF Upper(GetChar(['s','a','A','S'])) = 'S'
THEN Save
ELSE Append;
END
ELSE Append
END
ELSE WRITELN('File must be created first');
END;
'L' : BEGIN
IF FilePresent(ListFileName)
THEN
BEGIN
IF Modified
THEN
BEGIN

WRITE('OK to abandon changes');
IF YesNo = Yes
THEN Load
END
ELSE Load;
END
ELSE WRITELN('Must create file first');
END;

'Q' : IF Modified
THEN
BEGIN
WRITE('Ok to abandon changes to ', List.Name);
IF YesNo = No
THEN Choic := 'q';
END;
END;
UNTIL Choic = 'Q';
CLOSE(Device);
WRITELN;
WRITELN('END OF PROGRAM');
END.
The Byte Comparison Program was used when the MiniPas history files were examined. This program had two purposes. The first purpose was as the program name implies: to do a byte by byte comparison between adjacent versions of a given program stored in the history files. The second purpose was to locate the point in the program where the compiler encountered the syntax errors. Both of these tasks when manually done were tedious and error-prone; therefore, the use of this program enhanced the reliability of examining the programming protocols.

A listing of the Byte Comparison is provided below. The program was written in Turbo Pascal running on a DEC Rainbow 100 microcomputer. The UTILITY.PAS file which was included by the compiler directive, (*$I Utility.PAS*), is provided in Appendix R.

(*$c-,u-*)

PROGRAM CompareBytes(INPUT, OUTPUT);

(*$I Utility.PAS *)

CONST
  Debug = FALSE;
  Max = 750;
  Box1 = 1; (* crt line of first line of box 1 *)
  Box2 = 11; (* crt line of first line of box 2 *)
  Height = 9; (* height of both boxes *)

TYPE
  ListType = ARRAY[1..Max] OF Str80;
  Control = (UpCursor, DownCursor, LeftCursor, RightCursor, Return, UpArrow, DownArrow, LeftArrow, RightArrow, Start, EndX, Up, Down, Top, Bottom, ForwardX, Backward, Mark, Jump, Find, Locate, Position);

VAR
Loaded, JustCompared : BOOLEAN;
Option : CHAR;
Filename : Str80;
Size,X1,Y1,Top1,Bot1,Oldx1,Oldy1,
X2,Y2,Top2,Bot2,Oldx2,Oldy2 : INTEGER;
List : ListType;

PROCEDURE Prompt(Line : INTEGER;
Which : CHAR;
Msg : Str80);
BEGIN
GOTOXY(0,Line);ClrEOL;
GOTOXY(0,Line);
IF Line=22
THEN
CASE Which OF
'M' : BEGIN
WRITE('MAIN MENU: Kbox 2(box C(ompare');
WRITE(' S(etup B(ackward L(oad Q(uit');
END;
'1' : BEGIN
WRITE('BOX1: <RET> UDSETB, F(ind,J(ump,');
WRITE('L(ocate,P(osition,M(arke');
END;
'2' : BEGIN
WRITE('BOX2: <RET> UDSETB, F(ind,J(ump,');
WRITE('L(ocate,P(osition,M(arke');
END;
'C' : WRITE('Comparing...');
'S' : WRITE('Setting up...');
'*' : WRITE(Msg);
END
ELSE
CASE Which OF
'L' : WRITE('Locating compiler error location...');
'R' : WRITE('File too long, only first 'max,' lines loaded');
'*' : WRITE(Msg);
END
END;

PROCEDURE ReadList(Filename : Str80);
VAR
EndOfFile : BOOLEAN;
S : STRING[255];
T : TEXT;
BEGIN
Assign(T,Filename);
RESET(T);
Size := 0;

EndOfFile := EOF(T);
WHILE NOT EndOfFile AND (Size < Max) DO
BEGIN
  Size := Size + 1;
  READLN(T,S);
  IF Length(S) = 0
  THEN List[Size] := ' '
  ELSE IF Length(S) > 76 (* 80 minus 4 for line numbers *)
  THEN List[Size] := Copy(S,1,76)
  ELSE List[Size] := S;
  EndOfFile := EOF(T);
END;
IF NOT EndOfFile AND (Size=Max)
THEN Prompt(23,'R',' ');
PPReturn;
CLOSE(T);
End;

FUNCTION GetArrow : Control;
VAR
  Key, Key2, Key3 : CHAR;
  Temp : Control;
BEGIN
  (*D-*)
  REPEAT
    READ(KBD,KEY);
    Key := Upper(Key);
    IF Key=CHR(27)
    THEN
      BEGIN
        READ(Kbd,Key2,Key3);
        CASE ORD(Key3) OF
        65 : Temp := UpArrow;
        66 : Temp := DownArrow;
        68 : Temp := LeftArrow;
        67 : Temp := RightArrow;
        END;
      END
    ELSE
      BEGIN
        Temp := Return;
        CASE Key OF
        'S' : Temp := Start;
        'E' : Temp := Endx;
        'U' : Temp := Up;
        'D' : Temp := Down;
        'T' : Temp := Top;
        'B' : Temp := Bottom;
      END
    END
  END
END
'M' : Temp := Mark;
'J' : Temp := Jump;
'F' : Temp := Find;
'P' : Temp := Position;
'L' : Temp := Locate;
'F' : Temp := Backward;
'.' : Temp := Forwardx;
END; (* case *)

UNTIL (Key=CHR(13)) OR
((Key=CHR(27)) AND (ORD(Key3) IN [65,66,67,68])) OR
(Key IN ['S','E','U','D','T','B','M','J','F','P','L', ''])

GetArrow := Temp;
(* $U+*)

END;

PROCEDURE MarkIt(CRTLine,X,Y,Topp : INTEGER);
BEGIN
GOTOXY(X+4,CRTLine+(Y-Topp));
END;

PROCEDURE StarIt(CRTLine,X,Y,Topp : INTEGER);
BEGIN
GOTOXY(4,CRTLine+(Y-Topp));WRITE('@');
END;

PROCEDURE ShowXY;
BEGIN
GOTOXY(0,24); WRITE('Upper',Y1:4,X1:3,' Lower',Y2:4,X2:3);
END;

PROCEDURE Show(CRTLine, X,Y, Topp,Bott : INTEGER);
VAR
  I, Line, Leftover : INTEGER;
BEGIN
  Line := CRTLine;
  FOR I := Topp TO Bott DO
    BEGIN
      GOTOXY(0,Line);ClrEOL;
      GOTOXY(0,Line);WRITE(I:3,' ',List[I]);
      Line := Line + 1;
    END;
  Leftover := Height - Line + CRTLine + 1;
  FOR I := 1 TO Leftover DO
    BEGIN
      GOTOXY(1,Line);ClrEOL;
      Line := Line + 1;
    END;
END;
PROCEDURE ForceAdjust(CRTLine, X, Y : INTEGER;
                     VAR Topp, Bott : INTEGER);
BEGIN
  Topp := Y - Height DIV 2 + 1;
  InRange(Topp, 1, Y);
  Bott := Topp + Height - 1;
  InRange(Bott, Y, Size);
  Show(CRTLine, X, Y, Topp, Bott);
END;

PROCEDURE Adjust(CRTLine, X, Y : INTEGER;
                 VAR Topp, Bott : INTEGER);
BEGIN
  IF (Y >= Topp) AND (Y <= Bott) THEN MarkIt(CRTLine, X, Y, Topp)
  ELSE ForceAdjust(CRTLine, X, Y, Topp, Bott);
END;

PROCEDURE Setup(CRTLine, Direction : INTEGER;
                VAR X, Y, Topp, Bott : INTEGER);
VAR
  Found : BOOLEAN;
  Line : INTEGER;
BEGIN
  InRange(Y, 3, Size);
  Line := Y - 1; Found := FALSE;
  WHILE (Line < Size) AND (Line > 1) AND NOT Found DO
    BEGIN
      Line := Line + Direction;
      Found := (Pos('Compile:', List[Line]) = 1) OR
                (Pos('Run:', List[Line]) = 1);
    END;
  IF Found THEN
    BEGIN
      Y := Line + 1; X := 1;
      Adjust(CRTLine, X, Y, Topp, Bott);
      StarIt(CRTLine, X, Y, Topp);
      END;
END;
PROCEDURE LocCursor(CRTLine : INTEGER;
    VAR X, Y, Topp, Bott, OldX, OldY : INTEGER);

VAR
    Found, Edge : BOOLEAN;
    Line, Dummy : INTEGER;
    Item, S : Str80;
    Key : Control;
BEGIN
    InRange(Y, 1, Size);
    InRange(X, 1, Length(List[Y]));
    Adjust(CRTLine, X, Y, Topp, Bott);
    REPEAT
        GOTOXY(50, 24);
        WRITE('Topp=', Topp:3, ' Bott=', Bott:3, ' X=', X:3, ' Y=', Y:3);
        MarkIt(CRTLine, X, Y, Topp);
        Key := GetArrow;
        CASE Key OF
            UpArrow : BEGIN
                Edge := (Y = Topp(*+1*)) AND (Topp <> 1);
                IF Y > 1 THEN Y := Y - 1;
                IF Edge THEN Adjust(CRTLine, X, Y, Topp, Bott);
                END;
            DownArrow : BEGIN
                Edge := (Y = Bott(*-1*)) AND (Bott <> Size);
                IF Y < Size THEN Y := Y + 1;
                IF Edge THEN Adjust(CRTLine, X, Y, Topp, Bott);
                END;
            RightArrow : BEGIN
                X := X + 1;
                IF (X > Length(List[Y])) AND (Y < Size)
                    THEN
                    BEGIN
                        X := 1;
                        Y := Y + 1;
                    END
                    ELSE
                        InRange(X, 1, Length(List[Y]));
                    END;
            LeftArrow : BEGIN
                X := X - 1;
                IF (X < 1) AND (Y > Topp)
                    THEN
                    BEGIN
                        Y := Y - 1;
                        X := Length(List[Y]);
                    END
                    ELSE
                        InRange(X, 1, Length(List[Y]));
                    END;
            Start : X := 1;
EndX  : X := Length(List[Y]);
Top   : BEGIN
Y := 1;
X := 1;
Adjust(CRTLine,X,Y,Topp,Bott);
END;
Bottom : BEGIN
Y := Size;
X := Length(List[Y]);
Adjust(CRTLine,X,Y,Topp,Bott);
END;
Up    : BEGIN
Y := Y - Height;
InRange(Y,1,Y);
Adjust(CRTLine,X,Y,Topp,Bott);
END;
Down  : BEGIN
Y := Y + Height;
InRange(Y,Y,Size);
Adjust(CRTLine,X,Y,Topp,Bott);
END;
Backward : Setup(CRTLine,-1, X,Y, Topp,Bott);
Forward  : Setup(CRTLine,+1, X,Y, Topp,Bott);
Find   : BEGIN
OldY := Y; OldX := X;
GOTOXY(0,23);ClrEOL;
GOTOXY(0,23);WRITE('Find: ');
READLN(Item);
Line := Y; Found := FALSE;
WHILE (Line < Size) AND NOT Found DO
BEGIN
Line := Line + 1;
Found := Pos(Item,List[Line]) <> 0;
END;
IF Found THEN
BEGIN
Y := Line;
Adjust(CRTLine,X,Y,Topp,Bott);
END
ELSE Prompt(23,'*', Item+' not found');
END;
Locate : BEGIN
Prompt(23,'L','');
OldX := X; OldY := Y;
IF Pos('Line: ', List[Y]) = 0
THEN Prompt(23,'*','Cursor must be on the compiler',
' error message line. ')
ELSE
BEGIN
S := Copy(List[Y], Pos('Column: ', List[Y])+8, 2); Val(S, X, Dummy); S := Copy(List[Y], Pos('Line: ', List[Y])+6, 2); Val(S, Y, Dummy); Y := Y + OldY; Adjust(CRTLine, X, Y, Topp, Bott); Prompt(23, '*','Cursor marks compiler error', ' location.'); OldY := OldY + 1; OldX := 1; END;

Position: Begin
X := OldX; Y := OldY; Adjust(CRTLine, X, Y, Topp, Bott); Prompt(23, '*','Cursor positioned at old position'); END;

Jump : BEGIN
OldX := X; OldY := Y;
Prompt(23, '*','Jump Line: ');
READLN(Y);
InRange(Y, 1, Size);
X := 1;
Adjust(CRTLine, X, Y, Topp, Bott);
GOTOXY(0, 23); ClrEOL;
END;

Mark : BEGIN
OldX := X; OldY := Y;
END; (* case *)
ShowXY;
MarkIt(CRTLine, X, Y, Topp);
UNTIL Key = Return;
END;

(* End Of Line *)
FUNCTION EOL(X, Y : INTEGER) : BOOLEAN;
BEGIN
EOL := (X=Length(List[Y]));
END;

(* End Of Array *)
FUNCTION EOA(X, Y : INTEGER) : BOOLEAN;
BEGIN
EOA := EOL(X, Y) AND (Y=Size);
END;

PROCEDURE Kick(VAR X, Y : INTEGER);
BEGIN
IF NOT EOA(X, Y)
THEN
  IF EOL(X,Y)
  THEN BEGIN X:=1; Y:=Y+1; END
  ELSE X:=X+1;
END;

PROCEDURE Compare;
VAR
  Diff : BOOLEAN;
BEGIN
  Diff := FALSE;
  WHILE NOT EOA(X2,Y2) AND NOT Diff DO
    BEGIN
      Diff := List[Y1][X1] <> List[Y2][X2];
      IF NOT Diff
        THEN BEGIN Kick(X1,Y1); Kick(X2,Y2); END;
    END;
  Adjust( l,X1,Y1,Top1,Bot1);
  Adjust(l1,X2,Y2,Top2,Bot2);
  IF Diff
    THEN
      BEGIN
        StarIt(l,X1,Y1,Top1);
        StarIt(l1,X2,Y2,Top2);
        Prompt(23,'*','difference is marked')
      END
    ELSE Prompt(23,'*','end of array');
END;

PROCEDURE Load(FileName : Str80);
BEGIN
  ReadList(FileName);
  Loaded := TRUE;
  X1 := 1; Y1 := 1; Top1 := 1; Bot1 := Height;
  X2 := 1; Y2 := 1; Top2 := 1; Bot2 := Height;
  InRange(Bot1,1,Size);
  InRange(Bot2,1,Size);
  Show( l,X1,Y1,Top1,Bot1);
  Show(l1,X2,Y2,Top2,Bot2)
END;

BEGIN
  InitUtility;
  SetUpper;
  ClrSCR;
  JustCompared := FALSE;
END;
Prompt(22,'M', '');
IF JustCompared
THEN MarkIt(l1,X2,Y2,Top2);
Option := Choice('12CSBLQ');
JustCompared := FALSE;
Prompt(22,Option, '');
IF Loaded OR (Option='L')
THEN
CASE Option OF
'1' : LocCursor(l, X1,Y1, Top1,Bot1, Oldx1,Oldy1);
'2' : LocCursor(l1, X2,Y2, Top2,Bot2, Oldx2,Oldy2);
'C' : BEGIN
IF (X1=X2) AND (Y1=Y2)
THEN Prompt(23,'*','Both cursor positions are the same,','
' trival to compare')
ELSE
BEGIN
Compare;
Prompt(23,'*','Difference is marked');
ShowXY;
JustCompared := TRUE;
END;
END;
'S' : BEGIN
Prompt(23,'*','Setting up forward...');
SetUp(l,l,X1,Y1,Top1,Bot1);
SetUp(l1,l,X2,Y2,Top2,Bot2);
ShowXY;
END;
'B' : BEGIN
Prompt(23,'*','Setting up backward...');
SetUp(l,-l,X1,Y1,Top1,Bot1);
SetUp(l1,-l,X2,Y2,Top2,Bot2);
ShowXY;
END;
'L' : BEGIN
Prompt(22,'*','File to load: ');
READLN(FileName);
IF FilePresent(FileName)
THEN
BEGIN
Prompt(23,'*','Loading...');
Load(FileName);
Prompt(23,'*',' ');
END
ELSE Prompt(23,'*','FileName + ' not found');
END;
END (* case *)
ELSE Prompt(23, '*','No file loaded');
UNTIL Option = 'Q';

END.
APPENDIX O. FISHER'S EXACT TEST PROGRAM

The chi-square or the t-test procedures are commonly used to test whether or not two population proportions are equal when only nominal data are available. These two procedures are invalid when the expected cell frequencies are less than five. The Fisher's Exact Test can be used instead.

The basic formula for the Fisher's Exact Test is provided in Figure O-1. The result of computing this formula is a probability.

\[
\text{Fisher's Exact Probability} = \frac{A \times B}{(A + C) \times (B + D)}
\]

for a 2 by 2 Table

\[
\frac{A + B}{A + B + C + D}
\]

Where:
- A is frequency of upper left cell, number of successes for Group One
- B is frequency of upper right cell, number of successes for Group Two
- C is frequency of lower left cell, number of failures for Group One
- D is frequency of lower right cell, number of failures for Group Two

Figure O-1. Formula for Fisher's Exact Probability

This is interpreted as the probability of obtaining the given combination of frequencies in a 2 by 2 table. Since the test of interest is determine whether or not the difference between two proportions is statistically significant, all cases which are more extreme must also be calculated. More extreme cases are obtained by moving observations out of the cell containing the lowest frequency count into an adjacent cell (horizontally or vertically adjacent, not
diagonally adjacent). An observation must also be moved out of the cell which is diagonally opposite the cell with the lowest count and into a horizontally or vertically adjacent cell in order to obtain the original row and column totals. The sum of the original probability and the probabilities of all more extreme cases are summed together to obtain the overall probability.

The Fisher's Exact Test requires the calculation of several factorials. The worst case factorial involved in the calculations is factorial \( n \); where, \( n \) is the number of cases. To minimize round-off errors and prevent an numeric overflow error, two programming adjustments were made. The first adjustment was to use the binary coded decimal (BCD) extension of Turbo Pascal for real variables. The BCD reals in Turbo Pascal maintains 18 significant decimal digits. The second adjustment was to optimize (or reduce) the number of multiplications required in calculating the factorials. Since factorials are found both in the numerator and denominator in several of the calculations, some of the terms can be canceled and thereby reducing the number of multiplications and thus reducing the number of multiplications and the number of round off errors which could accumulate. The largest factorial which can be calculated by this program is factorial 49 before there is an overflow error. A listing of the Fisher's Exact Test is provided below.

(* Needs to be run under BCD Turbo Pascal due to large factorial values which must be calculated *)

PROGRAM Fisher(INPUT, OUTPUT);

TYPE
  CellType = (UpLeft, UpRight, LoLeft, LoRight);
VAR
    Answer : CHAR;
    A,B,C,D,N,I : INTEGER;
    Probability, TotalP : REAL;
    Cell : CellType;

FUNCTION Min(x, y : INTEGER) : INTEGER;
BEGIN
    IF X < Y THEN Min := X; ELSE Min := Y; END;

FUNCTION Factorial(Start, N : INTEGER) : REAL;
VAR
    I : INTEGER;
    Temp : REAL;
BEGIN
    Temp := 1;
    FOR I := Start TO N DO Temp := Temp * I;
    Factorial := Temp;
END;

FUNCTION Choose(N, X : INTEGER) : REAL; (* as per standard formula *)
BEGIN
    Choose := Factorial(1,N) /(Factorial(1,x) * Factorial(1,N-X));
END;

(* Choose2 is arithmetically the same as Choose but common factors in the denominator and numerator are canceled to reduce number of calculations*)
FUNCTION Choose2(N,X : INTEGER) : REAL;
BEGIN
    Choose2 := Factorial(X+1,N) /Factorial(1,N-X);
END;

FUNCTION Fisher(A,B,C,D : INTEGER) : REAL;
BEGIN
    Fisher := Choose2(A+C,A) * Choose2(B+D,B) / Choose2(A+B+C+D,A+B);
END;

BEGIN
    WRITELN('Fisher' s Exact Test for a 2 x 2 Table');
    REPEAT
        WRITELN;

END
195

WRITELN('Enter cell frequencies');
WRITE('Upper left cell: ');
READLN(A);
WRITE('Upper right cell: ');
READLN(B);
WRITE('Lower left cell: ');
READLN(C);
WRITE('Lower right cell: ');
READLN(D);
Probability := Fisher(A,B,C,D);
TotalP := Probability;
WRITELN('P':6, 'TotalP':10);
WRITELN(Probability:6:4, TotalP:10:4);
IF Min(A,B) < Min(C,D) (* find row of cells with smallest count *)
THEN
  IF A < B (* smallest count in top row *)
  THEN Cell := UpLeft
  ELSE Cell := UpRight
ELSE
  IF C < D (* smallest count in bottom row *)
  THEN Cell := LoLeft
  ELSE Cell := LoRight;
IF Cell IN [UpLeft, LoRight]
THEN
  FOR I := 1 TO Min(A,D) DO
  BEGIN
    A := A - 1;
    B := B + 1;
    C := C + 1;
    D := D - 1;
    Probability := Fisher(A,B,C,D);
    TotalP := TotalP + Probability;
    WRITELN(Probability:6:4, TotalP:10:4);
  END
ELSE
  FOR I := 1 TO Min(B,C) DO
  BEGIN
    A := A + 1;
    B := B - 1;
    C := C - 1;
    D := D + 1;
    Probability := Fisher(A,B,C,D);
    TotalP := TotalP + Probability;
    WRITELN(Probability:6:4, TotalP:10:4);
  END;
WRITE('again? ');
READ(KBD,Answer);
WRITELN;
UNTIL (Answer='n') or (Answer='N');
END.
APPENDIX P. INCLASS EXAMINATION

The inclass examination which was used as the criterion measure in the regression analyses is provided in this appendix. The examination consisted of 12 multiple choice items and 3 short answer, open-ended items. Students placed their answers to the 12 multiple choice items on a computer answer sheet and the items were scored by the ISU Test and Evaluation Services. Students wrote the answers to the three short answer items in the test booklet.

All items were scored dichotomously: 1 for correct and 0 for incorrect. The criterion variable was the sum of all the dichotomously scored items. The total number of points possible was 24. Each of the 12 multiple choice items was worth one point. The remaining 3 items constituted the other 12 points.

The scale for each of the last three items was constructed in the following manner. For a given open-ended item, a list of all the mistakes which were committed by all the students taking the test was compiled. The students made 23 different errors on the first open-ended question, 11 errors on the second question, and 13 errors on the third question. Again, dichotomous scoring was used: if the student did not commit the respective mistake, the mistake was scored as a 1; and if the student did commit the mistake, that mistake was scored as a 0.

The Kuder-Richardson Formula #20 (KR#20) was used to estimate the reliability of the exam. In order to improve the test reliability, certain test items were discarded. The criterion for item elimination was threefold:
1. All items with an item variance of zero were eliminated

2. All items with a point-biserial correlations which negative or less than .20 were eliminated

3. Items with a majority of its phi coefficients which were negative were eliminated unless the elimination resulted in the KR20 dropping.

Refer to Chapter III, Description of Measures Employed for a complete discussion of the item elimination process.

After completing the item elimination process, 24 items remained. Ten of the 12 multiple choice items remained; 8 of the 23 mistakes for the first open-ended item remained; 4 of the 11 mistakes for the second open-ended item remained, and 2 of the 13 mistakes for the third open-ended item remained. Provided below are the original 59 items; they are numbered #1, #2, ..., #59. The 24 items which remained after the item elimination process are enumerated with Roman numerals, I, II, ..., XXIV.
Test Directions: Select the best choice and mark the appropriate place on the computer answer sheet. Place your name and SS# on answer sheet and black in the dots for both name and SS# with a soft LEAD pencil.

#1. Rename, copy, print, type, dir are examples of
A. operating system commands
B. directory file names
C. editing commands
D. compiler commands
E. none of the above

#2. If a given segment of code must be executed a fixed number of times (i.e., the number of times is known prior to the execution of the segment), which would be the best control structure to select to do the job?
A. IF THEN
B. FOR DO
C. WHILE DO
D. REPEAT UNTIL
E. CASE OF

#3. Will the following program segment ever stop?

\[ \begin{align*}
J &:= 0; \\
\text{WHILE} \ (J \not< 10) \ \text{DO} \ J &:= J + 3;
\end{align*} \]

A. No
B. Yes
C. can't tell
D. depends on the computer system
4. What are the final values of TOTAL and J after this program segment finishes execution?

III

(1) TOTAL := 0;
(2) J := 1;
(3) WHILE (J < 5) DO
(4) BEGIN
(5) TOTAL := TOTAL + J;
(6) J := J + 1;
(7) END;

A. TOTAL = 6, J = 6
B. TOTAL = 0, J = 1
C. TOTAL = 5, J = 5
D. TOTAL = 15, J = 6
E. TOTAL = 10, J = 5

5. Replace line 5 in problem 4 by TOTAL := TOTAL + TOTAL; What is the final value of TOTAL now?
A. 0  B. 4  C. 5  D. 16  E. 32

6. The following program has an error. It should print THE END after the numbers read in total more than 20, but it doesn't. What should line (5) be changed to so that it will?

PROGRAM TEST;
VAR
    NUM, S : INTEGER;
BEGIN
    (1) S := 0;
    (2) WHILE (S <= 20) DO
    (3) BEGIN
    (4) READLN(NUM);
    (5) NEW := NUM + S;
    (6) END;
    (7) WRITELN('THE END');
END.

A. S := NUM;
B. NUM := S;
C. S := S + NUM;
D. NUM := NUM + S;
E. READLN(NUM + S);
#7. What statement should be in a Pascal program in order to assign the expression

\[
\frac{\text{Sum} + \frac{\text{Count} - 7}{2}}{2 \times \text{Total} + 1} - \text{Mean}
\]

to the variable, Result?

A) Result := Sum + Count - 7 / 2 * Total + 1 - (Mean * Mean);
B) Result := (Sum + Count - 7) / (2 * Total + 1) - Mean * Mean;
C) Result := Sum + (Count - 7) / (2 * Total + 1) - Mean * Mean;
D) Result := Sum + (Count - 7) / (2 * Total + 1 - Mean * Mean);
E) Result := Sum + Count - 7 / ((2 * Total + 1) - (Mean * Mean));
F) none of the above

In problems 8 to 10, refer to the following program segment. (Yes, indentation rules were broken, but can you still figure it out!)

IF A > 5 THEN B := 3
ELSE IF B < 5 THEN A := 3
ELSE A := 5;

#8. If A = 4 and B = 5 before execution, what will be their values after?

A. A = 4, B = 3  B. A = 3, B = 5  C. A = 3, B = 3
D. A = 5, B = 5  E. A = 5, B = 3

#9. If A = 4 and B = 4 before execution, what will be their values after?

A. A = 5, B = 4  B. A = 4, B = 3  C. A = 3, B = 4
D. A = 4, B = 4  E. A = 5, B = 3

#10. If A = 6 and B = 4 before execution, what will be their values after?

A. A = 6, B = 3  B. A = 5, B = 4  C. A = -3, B = 4
D. A = 6, B = 4  E. A = 5, B = 3
Use the following declarations for the next question:

VAR
    IntNum : INTEGER;
    RealNum : REAL;
    Ch : Char;

#11. Consider the assignment statements below:

   1) IntNum := RealNum;
   2) RealNum := IntNum;
   3) Ch := IF Option = 'N';
   4) IntNum := 10 DIV 5 / 2;
   5) REPEAT UNTIL TRUE;

Which of the assignment statements above are valid?

A) 2, 4, and 5 only
B) 1, 2, and 3 only
C) 1, 2, and 4 only
D) 2 and 5 only
E) only 2
F) none of them

#12. Which of the following are legitimate Pascal identifiers?

i) DrawBorder ii) Begin iii) MoveTo iv) 2ndTime

A) only (ii)
B) all but (ii)
C) only (i)
D) all four
E) (i) & (iii)
F) none of these
First Open-Ended Item. Given the following algebraic equation, write the equivalent Pascal assignment statement.

\[
(T + B^2) / 6 \\
H = \frac{X - Y}{K}
\]

List of errors made by students for first open-ended question

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XI</td>
<td>#13</td>
<td>used '=' instead of ':='</td>
</tr>
<tr>
<td>XII</td>
<td>#14</td>
<td>used superscript 2 instead of 'B*B'</td>
</tr>
<tr>
<td>XIII</td>
<td>#15</td>
<td>used implied multiplication</td>
</tr>
<tr>
<td>XIV</td>
<td>#16</td>
<td>no parentheses to force (X-Y)</td>
</tr>
<tr>
<td></td>
<td>#17</td>
<td>no parentheses to force (T+B*B)</td>
</tr>
<tr>
<td></td>
<td>#18</td>
<td>placed H on wrong side of :=</td>
</tr>
<tr>
<td></td>
<td>#19</td>
<td>used square or curly brackets instead of parentheses</td>
</tr>
<tr>
<td>XV</td>
<td>#20</td>
<td>'( ' &amp; ') ' not paired</td>
</tr>
<tr>
<td>XVI</td>
<td>#21</td>
<td>used IF THEN ELSE or other statement</td>
</tr>
<tr>
<td></td>
<td>#22</td>
<td>used FORTRAN '**' for raising to a power operator</td>
</tr>
<tr>
<td></td>
<td>#23</td>
<td>squared 'T+B' instead of just squaring 'B'</td>
</tr>
<tr>
<td></td>
<td>#24</td>
<td>divided by 'K' instead of multiplying by 'K'</td>
</tr>
<tr>
<td></td>
<td>#25</td>
<td>used &quot;B squared&quot; vs 'B*B'</td>
</tr>
<tr>
<td>XVII</td>
<td>#26</td>
<td>used BASIC up arrow instead of 'B=B'</td>
</tr>
<tr>
<td>XVIII</td>
<td>#27</td>
<td>used dot for multiplication operation instead of '*'</td>
</tr>
<tr>
<td></td>
<td>#28</td>
<td>expression only, no H := for a complete assignment statement</td>
</tr>
<tr>
<td></td>
<td>#29</td>
<td>used WRITELN statement instead of assignment statement</td>
</tr>
<tr>
<td></td>
<td>#30</td>
<td>multiplied 'B<em>2' instead of 'B</em>B'</td>
</tr>
<tr>
<td></td>
<td>#31</td>
<td>broke into multiple assignment statements but no ';'</td>
</tr>
<tr>
<td></td>
<td>#32</td>
<td>one pair parentheses around wrong terms</td>
</tr>
<tr>
<td></td>
<td>#33</td>
<td>two pair parentheses around wrong terms</td>
</tr>
<tr>
<td></td>
<td>#34</td>
<td>three pair parentheses around wrong terms</td>
</tr>
<tr>
<td></td>
<td>#35</td>
<td>used divide by sign (dash with dot above and below) instead of '/'</td>
</tr>
</tbody>
</table>
Second Open-Ended Item  Complete the variable declarations so that this program will compile successfully.

PROGRAM Figure(INPUT, OUTPUT);
VAR

BEGIN
  WRITELN('Enter the last odometer reading');
  REA DLN( Last );
  WRITELN('Enter the current odometer reading');
  REA DLN( Curr );
  WRITELN('How many gallons of gas did you');
  WRITELN('just put in the tank?');
  REA DLN( Gallons );
  Distance := Curr - Last;
  MPG := Distance / Gallons;

END.

List of errors made by students for second open-ended question

- #36 missing comma
- #37 missing colon
- #38 missing semicolon
- #39 missed one variable
- #40 missed more than one variable
- #41 missing 'REAL'
- #42 used 'INTEGER' type instead of 'REAL' type
- #43 declared extra variable
- #44 used CHAR instead of REAL
- #45 wrote an English phrase
- #46 split into two parts
Third Open-Ended Item. Add a WRITELN statement at the end of the above program which will print the output shown below. The numbers may vary, depending on what values the user enters when running the program. Don't worry about upper and lower case letters.

You got 25.4 miles to the gallon and traveled 200 miles

List of errors made by students for second open-ended question

#47 extra single quote
#48 missing part of text string
#49 extra comma in WRITELN
#50 missing comma(s)
#51 missing left or right parentheses
#52 missing single quote
XXIII #53 comma inside single quote
#54 used two WRITELNs
XXIV #55 wrong text, "out in left field"
#56 omitted all single quotes
#57 forgot single quote around miles
#58 no Distance variable in WRITELN
#59 rewrote sentence
APPENDIX Q. INTER-ITEM CORRELATIONS AND POINT-BISERIAL CORRELATIONS FOR INCLASS EXAMINATION ITEMS USED FOR CRITERION MEASURE

The inclass examination was used as the regression analysis criterion variable. In order to increase the internal reliability of the criterion measure, certain items were eliminated. Refer to Chapter III for complete details concerning the item elimination process. The final point-biserial correlations and inter-item correlations for the 24 items which remained in the criterion measure are provided in the tables below.

Table Q-1. Item with Total Score Point-Biserial Correlations for Inclass Examination

<table>
<thead>
<tr>
<th>Item</th>
<th>$r_{pb}$</th>
<th>Item</th>
<th>$r_{pb}$</th>
<th>Item</th>
<th>$r_{pb}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.47</td>
<td>9</td>
<td>.48</td>
<td>17</td>
<td>.29</td>
</tr>
<tr>
<td>2</td>
<td>.69</td>
<td>10</td>
<td>.27</td>
<td>18</td>
<td>.66</td>
</tr>
<tr>
<td>3</td>
<td>.45</td>
<td>11</td>
<td>.41</td>
<td>19</td>
<td>.29</td>
</tr>
<tr>
<td>4</td>
<td>.78</td>
<td>12</td>
<td>.54</td>
<td>20</td>
<td>.66</td>
</tr>
<tr>
<td>5</td>
<td>.50</td>
<td>13</td>
<td>.25</td>
<td>21</td>
<td>.66</td>
</tr>
<tr>
<td>6</td>
<td>.20</td>
<td>14</td>
<td>.58</td>
<td>22</td>
<td>.58</td>
</tr>
<tr>
<td>7</td>
<td>.71</td>
<td>15</td>
<td>.29</td>
<td>23</td>
<td>.32</td>
</tr>
<tr>
<td>8</td>
<td>.33</td>
<td>16</td>
<td>.66</td>
<td>24</td>
<td>.66</td>
</tr>
</tbody>
</table>
Table Q-2. Inter-Item Correlations (Phi Coefficient) for Inclass Examination

<table>
<thead>
<tr>
<th>Items</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.32</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.21</td>
<td>.32</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.39</td>
<td>.84</td>
<td>.34</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-.07</td>
<td>.32</td>
<td>.08</td>
<td>.39</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.32</td>
<td>.38</td>
<td>-.10</td>
<td>.32</td>
<td>-.07</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.27</td>
<td>.73</td>
<td>.12</td>
<td>.76</td>
<td>.44</td>
<td>.35</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>.15</td>
<td>.42</td>
<td>.17</td>
<td>.32</td>
<td>.15</td>
<td>.38</td>
<td>.55</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>.28</td>
<td>.21</td>
<td>-.06</td>
<td>.04</td>
<td>.16</td>
<td>.13</td>
<td>.12</td>
<td>.07</td>
<td>.09</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>.02</td>
<td>.25</td>
<td>.00</td>
<td>.30</td>
<td>.16</td>
<td>.08</td>
<td>.35</td>
<td>-.06</td>
<td>.22</td>
<td>.02</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>.19</td>
<td>.29</td>
<td>.41</td>
<td>.34</td>
<td>.19</td>
<td>.08</td>
<td>.24</td>
<td>.04</td>
<td>.18</td>
<td>.20</td>
<td>.12</td>
<td>1.00</td>
</tr>
<tr>
<td>13</td>
<td>-.05</td>
<td>.00</td>
<td>-.02</td>
<td>-.05</td>
<td>-.05</td>
<td>-.07</td>
<td>-.02</td>
<td>-.18</td>
<td>.23</td>
<td>-.01</td>
<td>.35</td>
<td>.08</td>
</tr>
<tr>
<td>14</td>
<td>.05</td>
<td>.10</td>
<td>.49</td>
<td>.26</td>
<td>.26</td>
<td>-.05</td>
<td>.08</td>
<td>.10</td>
<td>.26</td>
<td>-.05</td>
<td>.20</td>
<td>.43</td>
</tr>
<tr>
<td>15</td>
<td>-.11</td>
<td>-.09</td>
<td>.10</td>
<td>.18</td>
<td>.46</td>
<td>-.03</td>
<td>.20</td>
<td>.23</td>
<td>-.05</td>
<td>-.04</td>
<td>-.12</td>
<td>.16</td>
</tr>
<tr>
<td>16</td>
<td>.32</td>
<td>.38</td>
<td>.23</td>
<td>.32</td>
<td>.32</td>
<td>-.02</td>
<td>.35</td>
<td>-.06</td>
<td>.12</td>
<td>.13</td>
<td>.28</td>
<td>.30</td>
</tr>
<tr>
<td>17</td>
<td>-.11</td>
<td>-.09</td>
<td>.10</td>
<td>.18</td>
<td>.46</td>
<td>-.03</td>
<td>.20</td>
<td>.23</td>
<td>-.05</td>
<td>-.04</td>
<td>-.12</td>
<td>.16</td>
</tr>
<tr>
<td>18</td>
<td>.32</td>
<td>.38</td>
<td>.23</td>
<td>.32</td>
<td>.32</td>
<td>-.02</td>
<td>.35</td>
<td>-.06</td>
<td>.12</td>
<td>.13</td>
<td>.28</td>
<td>.30</td>
</tr>
<tr>
<td>19</td>
<td>-.07</td>
<td>-.06</td>
<td>.23</td>
<td>.32</td>
<td>.32</td>
<td>-.02</td>
<td>-.07</td>
<td>.06</td>
<td>.12</td>
<td>-.18</td>
<td>-.08</td>
<td>.30</td>
</tr>
<tr>
<td>20</td>
<td>.32</td>
<td>.38</td>
<td>.23</td>
<td>.32</td>
<td>.32</td>
<td>-.02</td>
<td>.35</td>
<td>-.06</td>
<td>.12</td>
<td>.13</td>
<td>.28</td>
<td>.30</td>
</tr>
<tr>
<td>21</td>
<td>.32</td>
<td>.38</td>
<td>.23</td>
<td>.32</td>
<td>.32</td>
<td>-.02</td>
<td>.35</td>
<td>-.06</td>
<td>.12</td>
<td>.13</td>
<td>.28</td>
<td>.30</td>
</tr>
<tr>
<td>22</td>
<td>.46</td>
<td>.23</td>
<td>.10</td>
<td>.46</td>
<td>.18</td>
<td>-.03</td>
<td>.50</td>
<td>-.09</td>
<td>.18</td>
<td>-.04</td>
<td>.40</td>
<td>.16</td>
</tr>
<tr>
<td>23</td>
<td>.13</td>
<td>-.09</td>
<td>.10</td>
<td>.46</td>
<td>.18</td>
<td>-.03</td>
<td>.20</td>
<td>-.09</td>
<td>.18</td>
<td>-.26</td>
<td>.14</td>
<td>.16</td>
</tr>
<tr>
<td>24</td>
<td>.32</td>
<td>.38</td>
<td>.23</td>
<td>.32</td>
<td>.32</td>
<td>-.02</td>
<td>.35</td>
<td>-.06</td>
<td>.12</td>
<td>.13</td>
<td>.28</td>
<td>.30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Items</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>.29</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>-.10</td>
<td>.31</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>.35</td>
<td>.48</td>
<td>-.03</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>-.10</td>
<td>.31</td>
<td>1.00</td>
<td>-.03</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>.35</td>
<td>.48</td>
<td>-.03</td>
<td>1.00</td>
<td>-.03</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>-.07</td>
<td>.48</td>
<td>.70</td>
<td>-.02</td>
<td>.70</td>
<td>-.02</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>.35</td>
<td>.48</td>
<td>-.03</td>
<td>1.00</td>
<td>-.03</td>
<td>1.00</td>
<td>-.02</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>.35</td>
<td>.48</td>
<td>-.03</td>
<td>1.00</td>
<td>-.03</td>
<td>1.00</td>
<td>-.02</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>.20</td>
<td>.31</td>
<td>-.05</td>
<td>.70</td>
<td>-.05</td>
<td>.70</td>
<td>-.03</td>
<td>.70</td>
<td>.70</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>-.10</td>
<td>.31</td>
<td>.48</td>
<td>-.03</td>
<td>.48</td>
<td>-.03</td>
<td>.70</td>
<td>-.03</td>
<td>-.03</td>
<td>.48</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>.38</td>
<td>.48</td>
<td>-.03</td>
<td>1.00</td>
<td>-.03</td>
<td>1.00</td>
<td>-.02</td>
<td>1.00</td>
<td>1.00</td>
<td>.70</td>
<td>.30</td>
<td>1.00</td>
</tr>
</tbody>
</table>
APPENDIX R. UTILITY INCLUDE FILE

The Closure, Byte Comparison, and Fisher programs which are found
in Appendices M, N, and O used a compiler directive to include the
UTILITY.PAS file. The UTILITY.PAS file is listed below.

(* TURBO Pascal Utility Routines
    written on the DEC Rainbow *)

(* during debugging to enable ctrl-c interrupts, active the compiler
directive at the end of the getarrow function local to the
index function *)

CONST
    Yes = 'Y';
    No = 'N';
    CRTTop = 1;
    CRTRows = 24;
    Pi = 3.1415926535;

TYPE
    Str20 = STRING[20];
    Str80 = STRING[80];
    SetOfChar = SET OF CHAR;

VAR
    PrinterOnLine, Hardcopy, Printer, Messages,
        EchoX, AtBottom, CaseUpper : BOOLEAN;
    CRTLines : SET OF 1..24;
    Blanks : Str80;
    OkChars : SetOfChar;
    Device : TEXT;
    FFF : FILE;

FUNCTION Upper(C : CHAR) : CHAR;
BEGIN
    IF C IN ['a'..'z']
    THEN Upper := CHR(ORD(C)-32)
    ELSE Upper := C;
END;

FUNCTION Lower(C : CHAR) : CHAR;
BEGIN
    IF C IN ['A'..'Z']
THEN Lower := CHR(ORD(C)+32)
ELSE Lower := C;
END;

PROCEDURE SetCase(VAR S : Str80;
                   MakeUpperCase : BOOLEAN);
VAR
  C : INTEGER;
BEGIN
  IF MakeUpperCase
  THEN
    FOR C := 1 TO LENGTH(S) DO
      S[C] := Upper(S[C])
  ELSE
    FOR C := 1 TO LENGTH(S) DO
      S[C] := Lower(S[C])
  END;
PROCEDURE SetUpUpper;
BEGIN
  CaseUpper := TRUE;
END;
PROCEDURE SetUpLow;
BEGIN
  CaseUpper := FALSE;
END;
PROCEDURE SetEcho(E : BOOLEAN);
BEGIN
  EchoX := E;
END;

FUNCTION Echo : BOOLEAN;
BEGIN
  Echo := EchoX;
END;

PROCEDURE InRange(VAR Value : INTEGER;
                   Low, High : INTEGER);
BEGIN
  IF Value < Low
  THEN Value := Low
  ELSE
IF Value > High
    THEN Value := High;
END;

PROCEDURE Center(Title : StrSO;
    Line : INTEGER);
    BEGIN
InRange(Line, CRTTop, CRTRows);
GOTOXY(40 - LENGTH(Title) DIV 2, Line);
WRITE(Title);
IF Line <> CRTRows
THEN WRITELN;
END;

FUNCTION Choice(S : StrSO) : CHAR;
    VAR
        Found : BOOLEAN;
        Temp : CHAR;
        Len, Ptr : INTEGER;
    BEGIN
IF CaseUpper
    THEN SetCase(S,TRUE);
Found := FALSE;
Len := LENGTH(S);
REPEAT
    READ(KBD, Temp);
    IF CaseUpper
        THEN Temp := Upper(Temp);
        Ptr := 0;
    WHILE (Ptr < Len) AND NOT Found DO
        BEGIN
            Ptr := Ptr + 1;
            Found := (S[Ptr] = Temp);
        END;
UNTIL Found;
Choice := Temp;
END;

FUNCTION GetChar(OkSet : SetOfChar) : CHAR;
    VAR
        Key : CHAR;
    BEGIN
    REPEAT
        READ(KBD, Key);
        IF CaseUpper
            THEN Key := Upper(Key);
            UNTIL Key In OkSet;
IF (Key IN OkChars) AND Echo
THEN WRITE(Key);
GetChar := Key;
END;

PROCEDURE ReadInt(VAR I : INTEGER);
VAR
  Error : INTEGER;
  TempStr : Str20;
BEGIN
  READLN(TempStr);
  Val(TempStr, I, Error);
  WHILE Error <> 0 DO
  BEGIN
    WRITE('Invalid integer given, please re-enter: ');
    READLN(TempStr);
    Val(TempStr, I, Error);
  END;
END;

PROCEDURE ReadReal(VAR R : REAL);
VAR
  Error : INTEGER;
  TempStr : Str20;
BEGIN
  READLN(TempStr);
  Val(TempStr, R, Error);
  WHILE Error <> 0 DO
  BEGIN
    WRITE('Invalid real given, please re-enter: ');
    READLN(TempStr);
    Val(TempStr, R, Error);
  END;
END;

PROCEDURE ReadIRange(VAR I : INTEGER;
  Lo, Hi : INTEGER);
BEGIN
  WRITE('(',Lo,'-',Hi,') ');
  ReadInt(I);
  WHILE (I<Lo) OR (I>Hi) DO
  BEGIN
    WRITE('Number must be in the range ', Lo, ' to ', Hi, ': ');
    ReadInt(I);
  END;
END;
PROCEDURE PReturn;
VAR
  Key : CHAR;
BEGIN
  REPEAT
    READ(Kbd, Key);
  UNTIL Key = CHR(13);
END;

PROCEDURE PPReturn;
BEGIN
  WRITE(' Press <RETURN>');
  PReturn;
  WRITELN;
END;

PROCEDURE PPReturnC;
BEGIN
  WRITE('Press <RETURN> to continue. ');
  PReturn;
  WRITELN;
END;

FUNCTION YesNo : CHAR;
VAR
  OldEcho : BOOLEAN;
  Key : CHAR;
BEGIN
  OldEcho := Echo;
  SetEcho(False);
  WRITE(' (Y/N)? ');
  Key := Upper(GetChar(['Y', 'y', 'N', 'n']));
  IF Key = Yes
    THEN WRITELN('Yes')
  ELSE WRITELN('No ');
  SetEcho(OldEcho);
  YesNo := Key;
END;

PROCEDURE WBlanks(XLoc, YLoc, NumberOfBlanks : INTEGER);
BEGIN
  GOTOXY(XLoc, YLoc);
  WRITE(COPY(Blanks, 1, NumberOfBlanks));
END;
PROCEDURE EraseXY(x, y, xx, yy : INTEGER);
VAR
  Line, Width : INTEGER;
BEGIN
  X := ABS(X); Y := ABS(Y); Xx := ABS(Xx); Yy := ABS(Yy);
  InRange(X, 1, 80); InRange(Y, 1, CRTRows);
  InRange(Xx, X, 80); InRange(Yy, Y, CRTRows);
  Width := Xx - X + 1;
  FOR Line := Y TO Yy DO
    WBlanks(X, Line, Width);
END;

FUNCTION Index(XLoc, YLoc, NumLines, PmtLine : integer) ;
BEGIN
  Done, ErasePtr, PmtUsed : BOOLEAN;
  PmtlineX, PmtLineY : INTEGER;
  Pmt : Str80;
  (* local *) PROCEDURE DrawArrow;
  BEGIN
    G0T0XY(XLoc-4, YLoc + CurrLine - 1);
    WRITE(' <--');
  END;
  (* local *) PROCEDURE EraseArrow;
  BEGIN
    G0T0XY(XLoc-4, YLoc + CurrLine - 1);
    WRITE(' ');
  END;
  (* local *) Function GetArrow; TControl;
  VAR
    Key, Key2, Key3 : CHAR;
Temp : TControl;
BEGIN
(*$U-*
REPEAT
READ(Kbd,Key);
IF Key=CHR(27)
THEN
BEGIN
READ(Kbd,Key2,Key3);
CASE ORD(Key3) OF
  65 : Temp := UpArrow;
  66 : Temp := DownArrow;
  68 : Temp := LeftArrow;
  67 : Temp := RightArrow;
END;
ELSE Temp := Return;
END;
UNTIL (Key=CHR(13)) OR
((Key=CHR(27)) AND (ORD(Key3) IN [65,66,67,68]));
GetArrow := Temp;
(* $U+*)
END;

BEGIN (* main body of Index *)
ErasePtr := CurrLine < 0;
CurrLine := ABS(CurrLine);
InRange(NumLines, 1, 24);
InRange(CurrLine, 1, NumLines);
Pmt := '<arrows keys> to move pointer, <RETURN> to select';
PmtUsed := TRUE;
IF PmtLine IN CRTLines (* then print prompt line starting at
left edge of CRT*)
THEN
BEGIN
  PmtLineX := 1;
PmtLineY := PmtLine;
  GOTOXY(i,PmtLine); WRITE(Pmt);
END
ELSE
IF PmtLine*(-1) IN CRTLines (* then center prompt line *)
THEN
BEGIN
  PmtLine := ABS(PmtLine);
PmtLineX := 40 - LENGTH(Pmt) DIV 2;
PmtLineY := PmtLine;
  Center(Pmt, PmtLineY);
END
ELSE
  IF PmtLine > 99
  THEN

BEGIN
PmtLineX := PmtLine DIV 100;
InRange(PmtLineX, 1, 80-LENGTH(Pmt));
PmtLineY := PmtLine MOD 100;
InRange(PmtLineY, 1, 24);
GOTOXY(PmtLineX, PmtLineY); WRITE(Pmt);
END
ELSE PmtUsed := FALSE;
DrawArrow;
Done := FALSE;
REPEAT
CASE GetArrow OF
  DownArrow,
    RightArrow : BEGIN
      EraseArrow;
      CurrLine := CurrLine + 1;
      IF CurrLine > NumLines
      THEN CurrLine := 1;
      DrawArrow;
      END;
  UpArrow,
  LeftArrow : BEGIN
    EraseArrow;
    CurrLine := CurrLine - 1;
    IF CurrLine < 1
    THEN CurrLine := NumLines;
    DrawArrow;
    END;
  Return : Done := TRUE;
END;
UNTIL Done;
IF PmtUsed
THEN
  WBlanks(PmtLineX, PmtLineY, LENGTH(Pmt));
  IF ErasePtr
  THEN EraseArrow;
  Index := CurrLine;
END; (* Index *)

PROCEDURE WantHardCopy;
BEGIN
WRITELN;
IF HardCopy
THEN CLOSE(Device)
ELSE HardCopy := TRUE;
IF PrinterOnLine
THEN
BEGIN
WRITE('Do you want hardcopy output');

...
IF YesNo = Yes THEN BEGIN
  ASSIGN(Device, 'LST:');
  REWRITE(Device);
  WRITE(Device, CHR(13));
  Printer := TRUE;
END
ELSE BEGIN
  ASSIGN(Device, 'CON:');
  REWRITE(Device);
  Printer := FALSE;
  WRITELN(Device, CHR(12));
END END ELSE BEGIN
  ASSIGN(Device, 'CON:');
  REWRITE(Device);
  Printer := FALSE;
  WRITELN(Device, CHR(12));
END;

PROCEDURE QuitHardCopy;
BEGIN
  IF HardCopy THEN BEGIN
    BEGIN
      IF NOT Printer THEN PPReturnC;
      WRITELN(Device, CHR(12));
      CLOSE(Device);
      HardCopy := FALSE;
    END;
  END;
END;

PROCEDURE Print(S : Str80;
  Field,
  Spacing : INTEGER);
VAR
  Len : INTEGER;
BEGIN
  IF HardCopy THEN BEGIN
    Len := LENGTH(S);
InRange(Len, 0, Field);
S := COPY(S, 1, Len);
WRITE(Device, S, ' ', Field - Len + Spacing);
END;

PROCEDURE Pad(Str : Str80;
Size : INTEGER;
Ch,
WhichEnd : CHAR;
VAR NewStr : Str80);
VAR
  Len : INTEGER;
BEGIN
  Len := LENGTH(Str);
  InRange(Len, 0, Size);
  NewStr := Copy(Str, 1, Len);
  IF Upper(WhichEnd) = 'R'
  THEN
      WHILE Length(NewStr) < Size DO
          NewStr := NewStr + ' ' 
  ELSE
      IF Upper(WhichEnd) = 'L'
      THEN
          WHILE Length(NewStr) < Size DO
              NewStr := ' ' + NewStr;
  END;

FUNCTION FilePresent(FileName : Str80) : BOOLEAN;
VAR
  Present : BOOLEAN;
  F : FILE;
BEGIN
  ASSIGN(F, FileName);
  (*$i-≈*)
  RESET(F);
  (*$i++*)
  Present := (IOResult = 0);
  IF Present
  THEN CLOSE(F);
  FilePresent := Present;
END;

PROCEDURE Extension(VAR FileName : Str80; Temp : Str20);
BEGIN
  READLN(FileName);
  IF Pos('.', FileName) = 0
THEN Filename := FileName + Temp;
END;

FUNCTION Stop : BOOLEAN;
VAR
   Key : CHAR;
BEGIN
   Key := '?';
   IF KeyPressed
      THEN READ(Kbd,Key);
   Stop := Key = ' ';
END;

PROCEDURE InitUtility;
BEGIN
   Blanks := ' '; Blanks := Blanks + Blanks + Blanks + Blanks;
   CRTLines := [1..24];
   OkChars := [CHR(32)..CHR(126)];
   SetEcho(TRUE);
   SetUpLow;
   Printer := FALSE;
   HardCopy := FALSE;
   Assign(FFF,'LST:');
   REWRITE(FFF);
   PrinterOnLine := IOResult = 0;
   END;