Hall, William Earl, III

THE CLOZE PROCEDURE AND SOFTWARE COMPREHENSION

The Ohio State University

Ph.D. 1984

University Microfilms International

300 N. Zeeb 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. Other ______

University Microfilms International
THE CLOZE PROCEDURE AND SOFTWARE COMPREHENSION

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

William Earl Hall, III, B.A., M.S.

* * * * *

The Ohio State University
1984

Reading Committee:
Assoc. Prof. Stuart H. Zweben
Ass't. Prof. John S. Bourlay
Ass't. Prof. Kamesh Ramakrishna

Approved By:

Stuart Zweben
Advisor
Department of Computer
and Information Science
This dissertation is dedicated to my wife, Mary, from whom I have learned much.
ACKNOWLEDGMENTS

My sincere thanks go to my adviser, Dr. Stuart Zweben, for the valuable time and effort that he contributed to the making of this dissertation. I am especially appreciative of the high degree of professional standards to which he adheres. I would also like to thank the members of my Reading Committee, Dr. John Gourlay and Dr. Kamesh Ramakrishna. This research was supported in part by grant DAAB29-80-K-0061 from the U.S. Army Research Office.
December 21, 1954 . . . . . . Born - Birmingham, Alabama

1975 . . . . . . . . . . . . . . . . . B.A. in Mathematics, Birmingham-Southern College, Birmingham, Alabama

1975-1976 . . . . . . . . . . . . . Graduate Teaching Associate, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

1977 . . . . . . . . . . . . . . . . . M.S. in Computer and Information Science, The Ohio State University, Columbus, Ohio

1976-1977 . . . . . . . . . . . . . Undergraduate Coordinating Adviser, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

1977-1979 . . . . . . . . . . . . . Graduate Research Associate, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

1979-1982 . . . . . . . . . . . . . Graduate Teaching Associate, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

1982-1983 . . . . . . . . . . . . . Graduate Research Associate, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

1983-1984 . . . . . . . . . . . . . Graduate Administrative Associate, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio


Instructor for Office of Continuing Education, The Ohio State University, Columbus, Ohio
FIELDS OF STUDY

Major Field:
Software Engineering/Programming Languages.
Dr. Stuart Zweben and
Dr. William Buttelmann

Minor Fields:
Computer Architecture. Dr. M. T. Liu
Operating Systems. Dr. Sandra Mamrak
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>111</td>
</tr>
<tr>
<td>VITA</td>
<td>17</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>18</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION TO SOFTWARE COMPREHENSION AND ITS MEASUREMENT</td>
<td>1</td>
</tr>
<tr>
<td>Previous Research</td>
<td>5</td>
</tr>
<tr>
<td>Measure Evaluation</td>
<td>17</td>
</tr>
<tr>
<td>II. THE CLOZE PROCEDURE</td>
<td>25</td>
</tr>
<tr>
<td>Advantages of Cloze over Common Measures</td>
<td>25</td>
</tr>
<tr>
<td>Previous Research with the Cloze Procedure</td>
<td>27</td>
</tr>
<tr>
<td>The Need for More Research in the Software Domain</td>
<td>36</td>
</tr>
<tr>
<td>III. VALIDATING THE EXPERIMENTAL MATERIALS</td>
<td>39</td>
</tr>
<tr>
<td>Pilot Study 1</td>
<td>41</td>
</tr>
<tr>
<td>Pilot Study 2</td>
<td>49</td>
</tr>
<tr>
<td>IV. VALIDATING CLOZE TESTS IN THE SOFTWARE DOMAIN</td>
<td>56</td>
</tr>
<tr>
<td>Experiment 1</td>
<td>62</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>68</td>
</tr>
<tr>
<td>V. ANALYSIS OF THE EXPERIMENT 1 AND EXPERIMENT 2 CLOZE TEST CHARACTERICS</td>
<td>77</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>92</td>
</tr>
<tr>
<td>Experiment 4</td>
<td>98</td>
</tr>
<tr>
<td>Inter-Experimental Results</td>
<td>102</td>
</tr>
<tr>
<td>VI. SUMMARY AND CONCLUSIONS</td>
<td>107</td>
</tr>
<tr>
<td>Summary of Current Research</td>
<td>110</td>
</tr>
<tr>
<td>Experimental Conclusions</td>
<td>111</td>
</tr>
<tr>
<td>Summary of Contributions of Research</td>
<td>119</td>
</tr>
<tr>
<td>Future Research Suggestions</td>
<td>121</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>125</td>
</tr>
<tr>
<td>APPENDICES</td>
<td></td>
</tr>
<tr>
<td>A. Background Survey for All Pilot Studies and Experiments</td>
<td>131</td>
</tr>
<tr>
<td>B. General Instruction Sheet for Pilot Studies</td>
<td>133</td>
</tr>
<tr>
<td>C. Program Description for Pilot Study 1</td>
<td>134</td>
</tr>
<tr>
<td>D. Listing of Easy Version of Program for Pilot Study 1</td>
<td>136</td>
</tr>
<tr>
<td>E. Listing of Hard Version of Program for Pilot Study 1</td>
<td>140</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>F. Quiz Instruction Sheet</td>
<td>144</td>
</tr>
<tr>
<td>G. Quiz for Easy Version of Program for Pilot Study 1</td>
<td>145</td>
</tr>
<tr>
<td>H. Quiz for Hard Version of Program for Pilot Study 1</td>
<td>150</td>
</tr>
<tr>
<td>I. Post-Experimental Questionnaire for All Pilot Studies and Experiments</td>
<td>155</td>
</tr>
<tr>
<td>J. Means and Standard Deviations of Background Survey Responses and Post-Experimental Questionnaire Responses for Pilot Study 1</td>
<td>156</td>
</tr>
<tr>
<td>K. Correlations of Background Survey Responses and Post-Experimental Questionnaire Responses with Independent and Dependent Variables for Pilot Study 1</td>
<td>159</td>
</tr>
<tr>
<td>L. Consent Form for Volunteer Subjects</td>
<td>162</td>
</tr>
<tr>
<td>M. Program Description for Pilot Study 2</td>
<td>167</td>
</tr>
<tr>
<td>N. Listing of Easy Version of Program for Pilot Study 2</td>
<td>165</td>
</tr>
<tr>
<td>O. Listing of Hard Version of Program for Pilot Study 2</td>
<td>171</td>
</tr>
<tr>
<td>P. Quiz for Easy Version of Program for Pilot Study 2</td>
<td>176</td>
</tr>
<tr>
<td>Q. Quiz for Hard Version of Program for Pilot Study 2</td>
<td>181</td>
</tr>
<tr>
<td>R. Means and Standard Deviations of Background Survey Responses and Post-Experimental Questionnaire Responses for Pilot Study 2</td>
<td>186</td>
</tr>
<tr>
<td>S. Correlations of Background Survey Responses and Post-Experimental Questionnaire Responses with Independent and Dependent Variables for Pilot Study 2</td>
<td>189</td>
</tr>
<tr>
<td>T. Instruction Sheet for Cloze Experiments</td>
<td>192</td>
</tr>
<tr>
<td>U. Program Description for Cloze Experiments</td>
<td>195</td>
</tr>
<tr>
<td>V. Easy Program Version Cloze Test for Experiment 1</td>
<td>195</td>
</tr>
<tr>
<td>W. Hard Program Version Cloze Test for Experiment 1</td>
<td>200</td>
</tr>
<tr>
<td>X. Means and Standard Deviations of Background Survey Responses and Post-Experimental Questionnaire Responses for Experiment 1</td>
<td>205</td>
</tr>
<tr>
<td>Y. Correlations of Background Survey Responses and Post-Experimental Questionnaire Responses with Program Version and Synonymic Score for Experiment 1</td>
<td>208</td>
</tr>
<tr>
<td>Z. Easy Program Version Cloze Test for Experiment 2</td>
<td>211</td>
</tr>
<tr>
<td>AA. Hard Program Version Cloze Test for Experiment 2</td>
<td>216</td>
</tr>
<tr>
<td>BB. Means and Standard Deviations of Background Survey Responses and Post-Experimental Questionnaire Responses for Experiment 2</td>
<td>221</td>
</tr>
<tr>
<td>CC. Correlations of Background Survey Responses and Post-Experimental Questionnaire Responses with Independent and Dependent Variables for Experiment 2</td>
<td>224</td>
</tr>
</tbody>
</table>
DD. Listing of Program-Dependent Items in the Cloze Tests Used in Experiments 1 through 4 ................................................................. 227
EE. Listing of Parameter Version of Program for Experiments 3 and 4 ................................................................................................. 228
FF. Listing of Global Version of Program for Experiments 3 and 4 ................................................................................................. 233
GG. Parameter Program Cloze Test for Experiment 3 ............... 236
HH. Global Program Cloze Test for Experiment 3 ............... 242
II. Means and Standard Deviations of Background Survey Responses for Experiment 3 ........................................ 246
JJ. Correlations of Background Survey Responses with Independent and Dependent Variables for Experiment 3 ... 248
KK. Parameter Program Cloze Test for Experiment 4 ............... 250
LL. Global Program Cloze Test for Experiment 4 ............... 256
MM. Means and Standard Deviations of Background Survey Responses for Experiment 4 ........................................ 260
NN. Correlations of Background Survey Responses with Independent and Dependent Variables for Experiment 4 ... 262
<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Practical Ease of Several Software Comprehension Measures</td>
<td>20</td>
</tr>
<tr>
<td>2. Types of Reliability Applicable to Several Software Comprehension Measures</td>
<td>22</td>
</tr>
<tr>
<td>3. Pilot Study 1 Reliability Analysis</td>
<td>45</td>
</tr>
<tr>
<td>4. Pilot Study 1 t-test Analysis of Time and Score</td>
<td>46</td>
</tr>
<tr>
<td>5. Pilot Study 2 Reliability Analysis</td>
<td>51</td>
</tr>
<tr>
<td>6. Pilot Study 2 Reliability Analysis After Deletion from Conventional Program Quiz of Two Items Negatively Correlated with Overall Score</td>
<td>52</td>
</tr>
<tr>
<td>7. Pilot Study 2 t-test Analysis of Time and Score</td>
<td>53</td>
</tr>
<tr>
<td>8. Pilot Study 2 t-test Analysis of Score After Deletion from Conventional Program Quiz of Two Items Negatively Correlated with Overall Score</td>
<td>54</td>
</tr>
<tr>
<td>9. Experiment 1 Reliability Analysis</td>
<td>63</td>
</tr>
<tr>
<td>10. Experiment 1 Reliability Analysis After Deletion from Hard Program Cloze Test of Four Items Negatively Correlated with Overall Score</td>
<td>64</td>
</tr>
<tr>
<td>11. Experiment 1 t-test Analysis of Percent Score</td>
<td>65</td>
</tr>
<tr>
<td>12. Experiment 1 t-test Analysis of Percent Score After Deletion from Hard Program Cloze Test of Four Items Negatively Correlated with Overall Score</td>
<td>65</td>
</tr>
<tr>
<td>13. Experiment 1 t-test Analysis of Percent Score Ignoring Cloze Items in Variable Declaration Statements</td>
<td>69</td>
</tr>
<tr>
<td>14. Experiment 2 Reliability Analysis</td>
<td>71</td>
</tr>
<tr>
<td>15. Experiment 2 t-test Analysis of Percent Score</td>
<td>71</td>
</tr>
<tr>
<td>16. Proportion of Program-Dependent Cloze Items for Experiments 1 and 2 Cloze Tests</td>
<td>86</td>
</tr>
<tr>
<td>17. Difficulty of Different Categories of Cloze Items in Experiments 1 and 2</td>
<td>86</td>
</tr>
<tr>
<td>18. Increase Between Experiments 1 and 2 in Error Rates and Proportions of Program-Dependent Cloze Items</td>
<td>88</td>
</tr>
<tr>
<td>19. Experiment 3 Reliability Analysis</td>
<td>94</td>
</tr>
<tr>
<td>20. Experiment 3 t-test Analysis of Score and Time to Completion</td>
<td>95</td>
</tr>
</tbody>
</table>
21. Difficulty of Different Categories of Cloze Items in Experiment 3 ................................. 95
22. Experiment 4 Reliability Analysis ........................................................... 100
23. Experiment 4 t-test Analysis of Score and Time to Completion .................................. 101
24. Difficulty of Different Categories of Cloze Items in Experiment 4 ............................... 101
25. Increase Between Experiments 3 and 4 in Error Rates and Proportions of Program-Dependent Cloze Items ........................................... 103
26. Comparison of Regression Estimates and Average Percentage Scores .......................... 104
CHAPTER I

Introduction to Software Comprehension and its Measurement

There are primarily two groups concerned with the measurement of software comprehension: computer science educators and software engineers. Computer science educators, interested in tracking educational progress, use software comprehension metrics to assess the aptitudinal and performance characteristics of their students. Software engineers, interested in gaining insight into methods for improving software quality, use software comprehension metrics to assess the comprehensibility characteristics of software. Another, more basic use for software comprehensibility measurement is the improvement of the software comprehensibility measurement process itself. Because of the poorly understood nature of software comprehension, software engineers use the few software comprehension measurement techniques that are currently considered trustworthy to provide a basis from which to judge other, possibly more reliable, valid, or less costly software comprehension metrics that have yet to be proven trustworthy.

Computer science educators and software engineers are concerned with measuring software comprehensibility because comprehension is central to almost all software tasks that require mental processing. Software development, testing, debugging, and modification all require software comprehension to one degree or another. Thus, software that is difficult to comprehend is, in general, also difficult to develop, test, debug, and modify. These difficulties in turn contribute to the high cost of software. Although this crucial nature of software comprehension is not newly discovered, only in the last fifteen years or so has the comprehensibility of software become an important research issue. This importance stems from the growing incomprehensibility of software. The reason that software is increasingly more incomprehensible is simple: the problems being tackled via software are
themselves growing more difficult. The software solutions to the tough problems of AI-based "expert" systems, real-time communications networking, large-scale relational database systems, as well as many other areas, are composed of millions of lines of code, contain thousands of decision points, manipulate many and various types of data structures, and consist of hundreds of interconnected routines that must work together harmoniously. All of these factors (e.g., lengthy code, numerous decision points, etc.) are believed to reduce software comprehensibility.

The difficulty of a problem is often defined by the difficulty of its easiest known solution. Although the problems tackled via software may be growing in difficulty, one path to mitigating the difficulty of the problems is to look for more comprehensible solutions. Software engineers are confronted with many software development techniques (e.g., top-down design, modular decomposition) and principles (e.g., variable names should be mnemonic, indentation should indicate control flow, etc.) thought to lead to more comprehensible solutions. Unfortunately, no one set of development techniques and principles is guaranteed to produce software with an acceptable level of comprehensibility. A comprehensibility measure, at a minimum, allows the comparison of problem solutions that have been developed using different sets of techniques and principles so that, all else being equal, the less comprehensible solutions may be rejected for the more. Although this minimum benefit of a comprehensibility measure is desirable in itself, comprehensibility metrics may also provide other, more valuable benefits. Comprehensibility metrics may specify the location of troublesome code, warn when code has crossed a comprehensibility threshold, and even indicate what steps may be taken to correct code with poor comprehensibility. It should be noted, however, that most software comprehension measurement research is still at the stage of attempting to provide the minimum benefit of rank ordering the comprehensibility of different programs.
The most widely accepted measure of comprehension within the software domain is the "question-answer" quiz. A question-answer quiz requires subjects to either generate or recognize the correct answers to a set of questions that have been crafted by a test-maker to reveal comprehension of a piece of software. It is assumed by the test-maker that a subject's comprehension of the software is an increasing function of the number of questions the subject answers correctly. Although question-answer quizzes are generally regarded as a reliable and valid measure of software comprehension, they suffer from the drawbacks of requiring a great deal of effort and skill to develop.

A long-standing and proven measure of comprehension within the prose domain is the "cloze" test [R0BI81]. A prose-based cloze test requires subjects to fill-in words that have been deleted from a piece of text. It is assumed that a subject's comprehension of the original text is an increasing function of the number of words the subject replaces correctly. Cloze tests are known to be a reliable and valid measure of comprehension within the prose domain. However, unlike question-answer quizzes, cloze tests within the prose domain also have the attractive characteristic of requiring little effort or skill to develop. Recently, a few researchers have begun to use cloze tests within the software domain [ENTI84, COOK82, EHRLB2, N0RC82a]. These first few uses of software cloze tests provide some positive evidence of the efficacy of this easy-to-construct measure within this new domain. However, the widespread acceptance of the cloze test as a reliable and valid method for measuring software comprehensibility requires that much more experimental evidence favorable to the metric be gathered.

Because of the largely unknown nature of cloze tests in the software domain, the goals of this research effort were fundamental in character. A series of controlled human-subject experiments was conducted by the author to explore some of the theoretical and practical issues concerned with the use of cloze testing as a comprehensibility measure within the software domain. The reliability of software cloze tests was measured using standard statistical tests. The validity of cloze testing as a measure of software comprehensibility was studied by comparing the
results of rank ordering the comprehensibility of two programs using both cloze tests and the more accepted measure, question-answer quizzes. Other issues that were studied include the effects of scoring software cloze tests using different techniques as well as the effects of varying the percentages and types of missing parts within a cloze test.

The cloze tests used in the experiments proved to be easy to construct. They also were highly reliable. However, it was concluded from the experiments that software cloze tests constructed using standard techniques may lead to invalid results. A model for explaining these invalid results was constructed and validated. Several suggestions for changing the standard construction technique and thereby improving the validity of software cloze tests were derived from the model.

A proper understanding of the experiments requires a review of some background material. The remainder of this chapter is a short review of comprehension research, common software comprehension measurement tools, and methods for evaluating measures. The cloze "procedure" (as it is sometimes termed) is described in Chapter II. Advantages of the cloze procedure over the more common software comprehension measurement techniques are discussed using the bases of comparison established in Chapter I. A review is then made of the use of the cloze procedure by prose and software researchers. The need for more research of the cloze procedure within the software domain is also argued.

Two preliminary experiments (or pilot studies) are described in Chapter III. The pilot studies were performed to validate the use of materials that were used in subsequent experiments. The first two cloze experiments are presented in Chapter IV. These experiments were replications of a successful software cloze procedure experiment conducted by Cook et al. [COOK82]. A discrepancy in the results of the two replications prompted the author to develop a hypothesis concerning a possible flaw in the standard use of the cloze procedure. In Chapter V it is shown how the hypothesis explains the anomalous results of the previous chapter. The hypothesis is then used to develop and successfully predict the results of two more experiments. The
hypothesis is also used to explain the results of the original Cook experiment.

A summary of the author's research and the contributions of the research are presented in Chapter VI. Suggestions for future research are also contained in this concluding chapter.

Previous Research

A comprehensibility measure might suggest itself if one had an understanding of the inner workings of the comprehension process. Epistemologists have been reflecting on comprehension for centuries, but only more recently, with the advent of modern human-subject laboratory methods and the possibility of "artificial" comprehension by the computer, has much knowledge concerning comprehension been developed. In the next section we will briefly review the developments within the various disciplines attacking the problem of comprehension. Our focus will be on the field of cognitive psychology since practitioners within other disciplines tend to rely on psychologists to do the job of establishing the validity of their theories and models as applied to humans.

Models of Comprehension

Comprehension begins with the reception of physical energy; light waves for visual input, sound waves for aural input, etc. This energy is transmuted into "units" of meaning such as "ideas", "thoughts", and "beliefs". This input aspect of the comprehension process is observable. However, unless there is some corresponding form of observable output or resulting behavior, no comprehension can be known to have taken place. Although cognitive processes and structures, like input and output, are believed to have a physical basis, no one has seen a thought or watched a memory form. Information about cognitive processes and structures must be arrived at in an indirect manner. The assumed properties of cognitive processes and structures are inferred from the consequences of the cognitive processes and cognitive structures interacting with each other. Progress toward understanding
comprehension has been made through the testing of research hypotheses concerning the separate steps of the cognitive processes and the components of the cognitive structures. The research paradigm or model is often the conceptual source of these experimental hypotheses [LACH79].

The laboratory study of comprehension first started among psychologists. Bartlett [BART32] began investigating comprehension in 1932, but none of his contemporaries continued on with his work. Until recently few other psychologists have concerned themselves with comprehension. The reasons for this are manifold. To many, the study of comprehension seemed to be an intractable problem. As we have previously noted, comprehension is diffuse and intertwined with many other mental activities. Also, the processes responsible for comprehension are extraordinarily flexible and ill-defined. This makes it difficult for investigators to construct tightly researchable questions that can yield solid knowledge about comprehension. Beyond these difficulties is the problem that the concept of comprehension was too "mentalistic" for behaviorists and neobehaviorists whose research theories and models largely influenced psychology from 1930 to 1970 [LACH79]. Although much work has been done in the area of cognitive psychology in the last decade, it is not surprising that researchers still find themselves far from a firm understanding of comprehension.

Comprehension is generally studied from one of two perspectives, that of psycholinguistics or cognitive science. Both of these disciplines have in common the information processing paradigm espoused by Newell and Simon [NEWE75] that views the human as a general-purpose symbol manipulation system. Although the majority of the comprehension studies within both of these fields has been oriented toward prose comprehension, cognitive science researchers have also done a great deal of work on studying problem solving. This research has largely had the objective of developing and testing computer simulation theories based on protocols collected from individuals while solving a problem [NEWE72]. Because of conflicts in research methods, the need to learn
specialized computer languages, and other discipline-related preferences, problem solving never became a central research area for cognitive psychologists [LACH79].

The Psycholinguistic Approach to Comprehension

Psycholinguistics is a combination of the disciplines of psychology and linguistics and is concerned with examining the behavioral implications of linguistic theories. Psycholinguists were first influenced by linguist Noam Chomsky's transformational grammar theory [CHOM57] and his later "standard" or deep-structure/surface-structure theory [CHOM65] of language competence. Both of these theories were syntax oriented and stimulated much research into their implications for linguistic behavior. The interest of psycholinguists, though, began to shift away from syntax-based theories toward more semantics-based theories as they realized the great influence semantic considerations had on the outcomes of their syntax-oriented experiments. Currently psycholinguists are in the main concerned with studying coding, how linguistic input is converted into a mental representation; context, the informational environment which affects linguistic input; and previous knowledge, that which is known before new input. Most of the comprehension studies by psycholinguists are at the sentence level with research focused on narrowly defined issues [LACH79]. The attention of software engineers generally has been on studying more complex aspects of comprehension than those studied by psycholinguists.

The Cognitive Science Approach to Comprehension

Psycholinguistics is much more limited in its scope than the cognitive science research effort in understanding comprehension. Cognitive science researchers have attempted to construct "global models" of comprehension that encompass all aspects of human comprehension activities and are not specific to certain task-specific behaviors. The global models are too incomplete to be considered full-fledged theories or even true models, instead they are better thought of as "experiments in conceptualization" [LACH79, p. 462]. To the global modelers, "thought experiments" are as important to developing an adequate
comprehension theory as are laboratory experiments. Since theoretical thinking can proceed faster than experimental data collection, the global models have been changing at a fast rate. The rapidly shifting nature and inherent complexity of the global models tends to discourage laboratory validation of the models by cognitive psychologists [LACH79]. The lack of laboratory results does not hinder the cognitive science researchers from testing their theories on computers, but does discourage conclusions concerning human comprehension.

The Software Engineering Approach to Comprehension

Software engineers realize the importance of theories and models to help guide and unify research efforts. Shneiderman [SHNE80] has developed a cognitive model of programmer behavior that he has used to help formulate hypotheses concerning such program-oriented tasks as composition, comprehension, debugging, testing, modification, and learning. Shneiderman's model, known as the "syntactic/semantic" model, is similar to the Atkinson and Shiffrin "multistore" memory model [ATKI68] from cognitive psychology and its cousins [NDRM70, WAU65]. The software engineers Tracz [TRAC79], Brooks [BRO077], and Mayer [MAYE81] also make use of a model similar in structure to the Atkinson and Shiffrin multistore model.

The multistore memory models are typified by the conceptualization of memory being divided into two or three separate memories (or "stores"): a short-term and long-term memory, and (for some models) a sensory register. Stimulus input is received through the sensory register (or the external environment if no sensory register is included in the model) which is encoded in some form and sent through to the short-term memory. The short-term memory can receive information from both the sensory register (or the external environment) and the long-term memory. The short-term memory is thought to be the area of conscious information processing and is characterized as having a small capacity and as holding for only a short time information that is not under attention. The long-term memory only receives information from the short-term memory and can hold a practically unlimited amount of information for a
Multistore memory models were mostly tested by cognitive psychologists via simple memorization-recall tasks and memory duration tasks that had little concern for the meaning of the information processed during the experiment. Experimental data eventually showed that the different memory stores were not as differentiable from each other as was hoped. At the same time researchers were concluding that memory and other aspects of information processing should be investigated from a semantic instead of capacity/duration framework. The multistore models have lost their credibility among cognitive psychologists and are considered lacking in interesting (i.e., semantically oriented) explanatory power [LACH79, p. 260]. However, many popular software engineering principles which try to limit the amount of information that must be dealt with at one time (e.g., modularization, information hiding, top-down design, etc.), are at least in part supported by the notion that the small capacity of short-term memory can easily lead to "information overload".

Although researchers currently do not offer the student of software comprehension a full-blown and validated model of comprehension with all the details worked out, cognitive psychology is beyond considering comprehension as simply a series of inscrutable events. There are several general hypotheses concerning comprehension that may be derived from the vast amount of research that has taken place in cognitive psychology. These hypotheses are well enough accepted by the scientific community that they should be taken into account when new hypotheses concerning cognition are developed and new software comprehension experiments are performed.

One useful hypotheses is that cognition in general (or comprehension in particular) is the result of a series of mental events unfolding over time. The idea that "time is cognition" [LACH79, p. 133] suggests that time is a useful dependent variable when one is attempting to measure comprehension. More difficult information should take longer to comprehend than less difficult information. More difficult information should also be more open to error possibilities due to the extended
processing time needed to deal with it. Another useful notion is the speed-accuracy trade-off [FITI66]. People can sacrifice speed for accuracy and accuracy for speed when performing tasks. Where the balance is placed between speed and accuracy depends on motivation, bias, belief, and so on. Having speed (time) and accuracy both as dependent variables allows checking for data differences that may be due to factors influencing speed-accuracy trade-offs [WICK77].

Software engineers believe there are many factors that affect software comprehension: characteristics of the software itself, the nature of the problem solved by the software, knowledge brought to the comprehension task, the nature of the supporting documentation for the software, and others. Software engineers have largely concentrated their software comprehension research on the effects of characteristics of the software itself, to which we will limit our discussion. The software characteristics thought to affect software comprehension include almost all aspects of software: commenting [NORC8b, WOOD81, SHEP78], mnemonic variable names [CURT79a, WEIS74], indentation [MIAR83, CURT79b, SHNE76b, WEIS74], control structures [LOVE77b, LUCA74, WEIS74], cohesion [MYRE78], coupling [MYRE78], design [YOUR75], abstract data types [MEES79], modularization [WOOD81], program length [CURT79a], control flow [WEIS74], data flow [WEIS74], and more. A deceptively simple question of importance is: how may these factors be manipulated to increase software comprehensibility? Software comprehensibility measurement tools are needed to answer this question.

Common Software Comprehensibility Measurement Tools

We have already noted that comprehension may only be determined indirectly. It is through the behaviors of the comprehending agent that we must detect clues of understanding. This indirect nature of the detection of comprehension allows many different avenues of measurement. Since the total comprehension of any particular matter is impossible (as is the detection of all aspects of comprehension), no matter the approach, all measures of comprehension are incomplete estimates at best.
Comprehensibility measurement tools may be either "diagnostic" or "predictive". A diagnostic comprehensibility tool measures the comprehensibility of some material by assessing a subject's current comprehension of the material. A predictive tool measures comprehensibility by "forecasting" the future comprehension of the material by some subject. A potential advantage of a predictive comprehensibility tool over a diagnostic comprehensibility tool in the software domain is that the "forecast" need not be based on difficult to measure subject behavior and instead may depend on simple software characteristics.

Diagnostic Tools. There are several common diagnostic comprehensibility measurement tools used by researchers. A straightforward comprehension measurement technique is to ask questions of the comprehending agent. Proper responses to questions that probe the nature and significance of the various facets of the material to be understood generally indicate some comprehension of the material. This is especially true if the answers to the questions require inferences to be drawn from the material. Moher and Schneider (MDHE81, p. 226) suggest that questions concerning the comprehension of software cover the following categories:

1. **descriptive semantics** (e.g., "what does this procedure do?")
2. **program simulation** (e.g., "give the output for the following input" and vice versa)
3. **data structures** (e.g., "array X is dimensioned to 50 because...")
4. **program flow** (e.g., "statement 10 is executed if and only if...")
5. **program modification** (e.g., "if statement 10 is changed to ..., then how is the output affected?")

Comprehension questions may have either "fixed" responses such as with true-false and multiple-choice tests or "open" responses such as with fill-in-the-blank, short-answer, and essay tests. "Question-answer" comprehension tests are probably the most commonly used and widely accepted form of a software comprehension measure (e.g., SHNE82, WOOD81,
The most popular response type for question-answer tests is probably multiple-choice because of its ease of scoring.

Some researchers have used modification and debugging tasks to measure comprehensibility [see CURT79a and SHEP79]. Modification and debugging tasks are less direct diagnostic methods than "question-answer" quizzes for measuring comprehension due to the potentially large moderating effect of the tasks themselves on the experimental outcome. These two tools are assumed to test comprehension based on the notion that an understanding of the functionality of a program is necessary before any significant modification or correction. Debugging and modification tasks of sufficient difficulty should require an understanding of the specifications for the original program and the change to be performed (modification or correction) plus an understanding of their algorithms.

For the debugging task the specification of the correction must be extracted from the specification of the original program. All else being equal, debugging is probably a more difficult task than modification. The known presence of a bug makes all parts of the original program suspect and untrustworthy. When difficulties arise in the understanding of the algorithm of the program one cannot be certain if the difficulty stems from a mistake by the author of the program or from simply a knotty problem.

Other diagnostics methods of assessing comprehension are the techniques of producing a functionally equivalent version of a memorized program [e.g., CURT79a, LOVE77a, SHNE77, SHNE76a]. The conceptual basis for these techniques comes from the psycholinguistic notion that newly acquired semantic information remains longer in memory than does the corresponding syntactic information [SACH67]. One should better be able to reconstruct from memory a program that was understood than one which was not understood since the comprehended program supplies long-lasting semantic memories while the non-comprehended program only supplies short-lived syntactic memories.

Predictive Tools. GORD79 and CURT79a provide examples of the experimental usage of two common predictive measures of software
comprehensibility. One measure is Halstead’s E (programming effort) metric [HALS77] and the other is McCabe’s cyclomatic complexity measure, \( v(G) \) [MCCA76]. Although the E measure was designed as a gauge of the mental effort required to compose a program and the cyclomatic metric was designed as a computational complexity measure, it is assumed that they are both correlated with comprehensibility. Neither of the measures require a comprehending agent for their use since they are based solely on the syntactic structure of the program being measured.

**Measure Evaluation**

The many comprehensibility measures available presents the problem of how to select a quality measure. There are several theoretical and practical considerations that should be examined when investigating the "goodness" of a measure. The theoretical considerations have to do with the reliability and validity of the measure. The practical considerations have to do with the constructive, applicative, and interpretational ease of the measure.

**Theoretical Considerations**

**Reliability.** The reliability of a measure refers to its consistency. The same measure applied in a similar manner to the same material should result in a similar reading. An unreliable measure is of little use since one could not know when to trust the results of applying the measure without reference to still another measure.

Reliability is achieved by intra-subject consistency as well as by inter-subject variability. Reliability is formally defined in terms of "true" scores and observed scores. True scores are the parts of observed scores that are not affected by random error. Theoretically, the reliability of a measure is calculated by dividing the variance of a group’s true scores by the variance of the group’s observed scores [MEHR73]. The closer the true scores are to the observed scores, the higher the reliability of the measure. There are five basic types of reliability [ANAS68]: test-retest, alternate-form, split-half, Kuder-Richardson, and scorer. Operationally, reliability is estimated via a
correlational analysis. Not all types of reliability are applicable to all measures, but the more types of reliability a measure possesses, the better the measure is considered to be. Each of the five types of reliability is briefly discussed below.

(1) Test-retest reliability refers to consistency in results when a measure is applied several times with the same subjects under similar circumstances. The error variance between uses of a measure in a test-retest situation is that caused by the passage of time between applications of the measure. In general, test-retest reliability estimates may be spuriously high because of material remembered from previous applications of the measure. Attempting to mitigate the effects of remembered material by increasing the length of time between the applications of the measure may also produce problems. As the amount of time between the applications of the measure increases, the chances also increase that the subjects being measured may experience different rates of growth in the trait being measured. Such variable growth then makes the different applications of the measure incomparable. An obvious practical drawback with test-retest reliability is the inherent requirement that an experiment be conducted several times to apply the measure repeatedly.

(2) Alternate-form reliability refers to consistency in results when an equivalent form of a measure is applied several times with the same subjects under similar circumstances. The error variance in an alternate-form use of a measure is caused by the passage of time and by the changes made in the alternative forms measure. Besides the problem of demonstrating that two forms of a measure are equivalent, alternate-form reliability also suffers from the same drawbacks mentioned for test-retest reliability.

(3) Split-half reliability is concerned with the internal consistency of a measure that is composed of several items (e.g., a comprehension quiz with 15 questions). When the items are divided into two groups such as first and second halves or even-numbered
and odd-numbered halves and the scores of the two groups are similar, then the measure is said to have split-half reliability. If the scores of the two groups are dissimilar, then the measure may be an amalgamation of two or more different measures applicable to different domains. A major advantage of split-half reliability compared to test-retest and alternate-form reliability is that it only requires one application of the measure. However, a disadvantage is that a split-half reliability estimate is not, in general, unique since a test may be split in half many different ways. It is also possible that different splits may produce large fluctuations in the reliability estimate.

(4) Like split-half reliability, Kuder-Richardson reliability examines the inter-item consistency of a measure. The Kuder-Richardson formulas are based on the correlation between a test and a hypothetical equivalent test. The advantage of Kuder-Richardson over split-half is that Kuder-Richardson produces a unique value. Kuder-Richardson reliability may also reveal that a measure is a conglomeration and may be possibly subdivided into other, more reliable, measures.

(5) Scorer reliability refers to consistency in interpretation of measurement readings by different users of the measure. At a minimum a measure should have scorer reliability. Without scorer reliability the other types of reliability cannot be determined and the readings of a measure may not be trusted by its various users.

Mehrens [MEHR73] mentions six factors that may influence the reliability of a quiz or test:

(1) Test item homogeneity - In general, the more homogeneous the items on a test (i.e., the more the items all measure the same characteristic), the higher the reliability of the test as measured by internal consistency.

(2) Test length - Getting a larger sample of behavior by increasing the number of homogeneous items on a test also tends to increase the
reliability of the measure.

(3) Test item difficulty - Since test reliability is dependent on score variability in such a way that the lower the score variability, the lower the reliability, then score ceiling or floor effects decrease test reliability. Score ceiling effects can be caused by too easy a test or too much test time while score floor effects can be caused by too difficult a test or too little test time.

(4) Subject homogeneity - Subject homogeneity also tends to decrease score variability and, thus, tends to deflate test reliability.

(5) "Speed" - A pure speed test is one where each subject who reaches a test item may be expected to get it right, but there is not enough time for a subject to reach all the test items. Here score differences largely depend on the number of items completed. Because of the spuriously high interitem correlation that can be expected with a speed test, it is inappropriate to estimate the reliability of a speed test using internal consistency techniques.

(6) Scoring objectivity - The influence of scoring objectivity was implied in the previous discussion of scorer reliability. The more objective a method for scoring a test, the higher the reliability estimate as determined by inter-scorer correlation.

Guilford [GUIL54] also mentions that, in general, multiple-choice and other tests in which chance success is a factor have a lower reliability than tests in which chance is not a factor (such as completion items). The reason for this is that chance success makes test items appear easier than they actually are since the proportion of subjects correctly completing each item is often higher than the proportion of subjects who know the answer.

The Kuder-Richardson reliability estimate will be used to estimate the reliability of the measures used in the experiments conducted by the author. There are several formulas for calculating Kuder-Richardson reliability. The most commonly used formulas are known as KR-8, KR-14, KR-20, and KR-21. Each higher numbered formula is a simplified form (in
calculations required) of the lower numbered formulas. However, as the formula number increases, so do the number of assumptions made concerning the characteristics of the test items. Kuder and Richardson state in their original paper:

Reliabilities obtained from the formulas presented here are never overestimates. When the assumptions are rigidly fulfilled, the figures obtained are the exact values of test reliability as herein defined; if the assumptions are not met, the figures obtained are underestimates. [KUDE37, p. 159]

Specifically, formula KR-B will be used as the reliability estimate for the measures used in the author's experiments. The only assumption made by this formula is that each test item and the measure as a whole measure the same function [KUDE37, p. 155].

Guilford notes that "no hard-and-fast rules can be stated" concerning how high reliability coefficients should be before they may be considered acceptable [GUIL54, p. 388]. Mehrens mentions that "it is generally accepted that standardized tests used to assist in making decisions about individuals should have reliability coefficients of at least .85", but for judgments concerning groups, "a reliability coefficient of about .65 may suffice." [MEHR73, p. 122]. Guilford also notes that for "research purposes" (i.e., when one is exploring experimental hypotheses), "one can tolerate much lower reliabilities than one can for practical purposes of diagnosis and prediction" and that reliability coefficients of .50 have been accepted in this context [GUIL54, p. 388]. Since our choice of KR-B as the reliability estimate already results in a conservative estimate of a strong form of reliability (split-half), we will accept a reliability coefficient of .50 in this research. (Note that all reliability estimates may range up to 1.)

Validity. Validity refers to the degree to which a metric fulfills its function and actually measures what it purports to measure. The objective evaluation of the validity of a measure usually requires independent, external criteria of whatever the metric is designed to
measure (ANAS68). The three "types" of validity (Stan66), content, criterion-related, and construct, are distinguished by the techniques used to determine them. The techniques required by a particular measure are dependent on its use, but as with reliability, the more techniques that can be used to demonstrate validity, the better the measure is considered to be. It should be noted that reliability is the first step toward validity; a measure believed to be unreliable must also be considered invalid.

Content validity is concerned with whether a metric represents the behavior domain to be measured. For example, a measure of spelling ability should test the spelling of words instead of, say, testing vocabulary size. Content validity is the weakest form of validity and is mostly of concern for achievement tests (ANAS68). "Face" validity is similar to content validity, but is not validity in a technical sense. Face validity is concerned with whether a measure appears valid to untrained observers (ANAS68). Measures without face validity may be met with undue skepticism and resistance by subjects and thus be rendered ineffective.

Criterion-related validity is concerned with the effectiveness of a measure in determining a specific behavior of an individual in a specified situation (ANAS68). Criterion-related validity is established through the empirical investigation of the relationships of measure scores to other external data or criteria. Two types of criterion-related validity are concurrent and predictive validity (ANAS68). As one may easily conclude, predictive validity is of concern for measures used to forecast future outcomes and concurrent validity is of concern for measures used to diagnose the existing status of some variable.

Construct validity is similar to criterion-related validity, but has to do with the degree to which a metric is a measure of more broadly defined behaviors than those associated with criterion-related validity. Examples of such behaviors are intelligence and comprehension. These behaviors are not tied to easily identified criteria as is a narrowly defined behavior like spelling ability. The distinction between
construct validity and criterion-related validity is that criterion-related validity is established by comparing the measure in question to another measure of the same function. Since broadly defined behaviors are not, in general, defined by some simple criterion, construct validity is established by such techniques as demonstrating measurement differentiation by contrasted groups and correlating with other measures believed to be related to the behavior being measured [ANAS68].

Practical Considerations

Even if a measure is known to be theoretically sound (i.e., reliable and valid), it will probably be little used if it is not also easy to deal with. As previously stated, the practical dimensions on which a measure is evaluated are the ease of construction, application, and interpretation. Since what may be easy for one person may not be easy for another, there are no hard and fast rules for defining "ease".

A measure has constructive ease if little effort is required to construct, assemble, or otherwise implement the measure. A measure of software quality that requires one to write a 500,000 statement program for each programming language has little constructive ease. A measure has applicative ease if it easy to administer. A software complexity measure that requires a program to be evaluated by a panel of fifty programming experts is difficult to use and has little applicative ease. Finally, a measure has interpretive ease if the result of the application of the measure is easily translated into a comprehensible outcome. A measure that requires the scoring of 100 short-answer questions has less interpretive ease than a measure that requires the scoring of 100 true-false questions because of the difference in effort needed to transform the raw results into a comprehensible score. Also, a software reliability measure that produces a complex outcome such as a 5x5 matrix of scores on different dimensions is less easy to interpret than a measure that performs a ranking on a ten-point scale.
Evaluating the Common Measures

For the purposes of evaluating the common comprehensibility measures we have discussed (multiple-choice quiz; modification, debugging, and recall tasks; Halstead's effort measure, \( E \); and McCabe's cyclomatic measure, \( v(G) \)), we will use a common experimental scenario. Let us assume that we wish to examine the effects of a particular programming language feature on program comprehensibility. To do this we implement two versions of a program - one version with the language feature and one version without - and apply the measure to the two programs. The individual measures are compared to each other on the practical dimensions in Table 1. The measures are rated on a three point scale of "easy", "medium", and "hard".

Table 1

<table>
<thead>
<tr>
<th></th>
<th>Quiz</th>
<th>Mod</th>
<th>Debug</th>
<th>Recall</th>
<th>E</th>
<th>v(G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructive Ease</td>
<td>hard</td>
<td>med</td>
<td>hard</td>
<td>easy</td>
<td>easy</td>
<td>easy</td>
</tr>
<tr>
<td>Applicative Ease</td>
<td>med</td>
<td>har</td>
<td>hard</td>
<td>med</td>
<td>easy</td>
<td>easy</td>
</tr>
<tr>
<td>Interpretive Ease</td>
<td>easy</td>
<td>har</td>
<td>hard</td>
<td>har</td>
<td>easy</td>
<td>easy</td>
</tr>
</tbody>
</table>

No real effort is required to set up (construct) the recall task or the \( E \) and \( v(G) \) measures since no materials are required beyond the two programs to be compared. The modification task requires that one invent a realistic modification that can be completed in a reasonable amount of time and take into account the experimental language feature, but this is generally not too difficult. The debugging task requires that one "seed" each of the programs with a bug. It can take some ingenuity to seed each of the programs with a bug that is at the same time realistic, feasible to be corrected in a manageable time period, sensitive to the language feature being tested, and similar between the two programs. All these features of the bug are necessary for a properly designed experiment. There are also many difficulties in developing a proper
multiple-choice quiz. Each question should be of sufficient difficulty yet not too difficult to discriminate among subjects, the questions must be similar for the two programs, the content of the questions need to reflect the differences between the programs, the different responses for each question should all be plausible to a certain degree, as well as other difficulties.

The $E$ and $v(G)$ measures are extremely easy to apply. The source code of the two programs is all that is needed and application of the measures can be automated. The recall task and multiple-choice quiz are somewhat more difficult to use. Each measure requires the use of a sample of subjects, but the administration of the subjects is straightforward. The modification and debugging tasks also require a sample of subjects. To be realistic these tasks should allow the subjects to work in as normal a manner as possible while attempting alterations on the programs, i.e., the subjects should have access to compilers, editors, terminals, etc. Structuring the work environment in such a way that it familiar to the subjects and also experimentally controlled can be difficult. Of course one may simply design the modification and debugging tasks so that the subjects must do all their work "by hand" at a desk and not use a computer. This simplifies the application of the measure, but gives little feedback to the subjects about when they are finished with the task and makes the interpretation of the results more difficult.

The results of using the $E$ and $v(G)$ measures are easy to interpret. Each measure outputs a single number for each program that allows one to rank order the two versions. Multiple-choice tests are easy to score and time and a statistical analysis of the score and time data will also allow a rank ordering of the programs. The score to interpret for functional recall tasks is the amount of the program accurately recalled. This can be difficult to judge and requires a panel of experts to decide. The variable of interest for the modification and debugging tasks is the time to perform the required change. The accuracy of the change and perhaps the percentage of subjects to
complete the change are also possible variables if not enough time is allowed for most subjects to complete the change. The accuracy of a change is difficult to determine when only partially correct and requires a panel of experts.

A comparison of the theoretical soundness of the measures is more difficult to perform than the analysis of their practical usefulness. Only the E and v(G) measures are standardized in the sense that the exact same (or similar) measure is used by many researchers and with different software. The recall task is not a standard since there is no set method for scoring. There is also no one multiple-choice test, modification task, or debugging task that may be examined independent of the experimental application of the measure. Although some general theoretical statements can be made, a non-standardized measure may or may not be a reliable or valid comprehensibility measure for a specific program.

Not all types of reliability are applicable to all the measures. Table 2 shows which types of reliability are applicable for the various measures.

Table 2
Types of Reliability Applicable to Several Software Comprehension Measures

<table>
<thead>
<tr>
<th>Quiz</th>
<th>MOD</th>
<th>DEBUG</th>
<th>RECALL</th>
<th>E</th>
<th>v(G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST-RETEST</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>n/a</td>
</tr>
<tr>
<td>ALTERNATE-FORM</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>SPLIT-HALF</td>
<td>a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>KUDER-RICHARDSON</td>
<td>a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>SCORER</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

a = applicable
n/a = not applicable

Except for scorer reliability, software engineering researchers are seldom concerned with demonstrating the reliability of the comprehensibility measures that are used in their experiments. Their
aim simply is to use the measure and they assume that a "reasonably" constructed version of a common measure should be conceptually sound.

There need be no worry about the reliability of the $E$ and $v(G)$ measures since they are defined in a deterministic manner. The reliability of the other measures depends on their implementation. Because multiple-choice quizzes are open to examination by all the types of reliability, a specific multiple-choice test may be shown to be reliable with a high degree of confidence. Also, the multiple-choice quiz is the only measure of those discussed that may be tested for split-half and Kuder-Richardson reliability. Testing for these two types of reliability only requires one application of the measure. Except for scorer reliability, modification, debugging, and recall tasks can only be checked for reliability by expending the great effort of replicating a human-subject experiment.

As with reliability, software engineering researchers generally trust in the validity of the measures they use. In all the above cited comprehensibility experiments for each of the common measures no attempt was made to validate the measure of comprehension.

The validity of the standardized $E$ and $v(G)$ measures has not yet been established. Some software engineers even doubt the validity of the $E$ measure [MORA78] and it can be unreliable since different techniques used for calculating the metric can yield different results. Such doubt can dissuade further needed research. In considering the applicability of the $E$ and $v(G)$ measures, Curtis et al. conclude the metrics "may not represent the most important constructs for predicting the performance of experienced programmers" [CURT79a, p. 103]. Experienced programmers are the group that many software engineers are most interested in studying. Although the predictive measures $E$ and $v(G)$ may be the easiest to use of the measures discussed (since they do not require the use of human subjects), human-subject studies and the non-standardized diagnostic measures are still needed.

The validity of the non-standardized measures cannot be judged independent of a particular application, but some general statements may
be made. For one, the content of multiple-choice quizzes can be manipulated to make them sensitive to different levels of software comprehension such as low level statement-oriented comprehension and high level module-oriented comprehension. Further, questions may be selected to test for highly specific areas of comprehension such as certain aspects of a particular data structure. It is not clear what levels of comprehension are tested by modification and debugging tasks. It also seems that multiple-choice tests may be valid for all lengths of programs. Although Shneiderman defends the validity of recall tasks and sees "no limit to the size of the program that can be committed to memory" [SHNEBO, p. 32], it does not seem likely that recall tasks are valid for measuring the comprehensibility of large programs. One might expect that the effort of memorizing a large program soon becomes too great of a burden and overshadows the effect of the comprehensibility of the program.
CHAPTER II

The Cloze Procedure

The cloze test is a text comprehensibility measurement tool that has been in use by prose researchers for some thirty years. The word "cloze" was coined from "closure", a Gestalt psychology term referring to the human tendency to mentally complete a familiar but not-quite-finished pattern [TAYL53]. Cloze tests are constructed by replacing a percentage of words in a text with short blank lines. Subjects attempt to reconstruct the original text by filling-in the blank lines with the missing words. The more words correctly filled-in, the more comprehensible the text is assumed to be.

Computer scientists may readily notice a close relationship between a cloze test and an information science entropy measure [see MCLE70, TAYL54]. When the entropy (or unpredictability) of the correct response for each blank is at its minimum (zero), then cloze scores should be at their maximum (100%). Alternately, when the entropy of each blank is at its maximum, then cloze scores should be at their minimum.

Advantages of Cloze over Common Measures

Experience by prose researchers suggests that cloze tests potentially have four main advantages over the common concurrent (i.e., non-predictive) software comprehensibility measures (multiple-choice quizzes, modification tasks, debugging tasks, and recall tasks). A primary advantage is that, unlike these metrics, cloze tests can be standardized such that the same measure is used by many researchers with many different materials. The benefits of using a standardized measure include increased ease in replicating and comparing experiments and providing a common basis for experimentation and communication among

*There is no claim, though that the closure phenomenon is at work when a subject is "completing" a cloze test.
researchers. We will describe a standard cloze test in the next section.

Another primary advantage is that, unlike question/answer quizzes, modification tasks, and debugging tasks, cloze tests are largely independent of influences by materials extraneous to the program being measured. Cloze tests measure "comprehension in process, not comprehension as a product after the fact" [RANK78, p. 151, (emphasis his)]. When measuring the comprehensibility of a program, it is advantageous to avoid confounding effects such as skill in understanding questions or modification specifications and the interaction of program and bug characteristics.

Cloze tests also have a utilitarian advantage over the common concurrent measures. Cloze tests are much easier to construct than multiple-choice quizzes; much easier to construct, administer, and score than modification and debugging tasks; and much easier to score than recall tasks. We have previously mentioned that the construction of a multiple-choice quiz may easily take days and that the scoring of a recall task generally requires a panel of experts. In contrast, cloze tests can be constructed in a few hours and can be scored easily by one person. The construction and scoring of cloze tests is also potentially automatable, which should be of particular interest to computer scientists.

A fourth advantage is that, in general, cloze tests should be more reliable than other multi-item comprehension measures such as question-answer quizzes. This is because it is usually possible to include a larger number of items on a cloze test that may be completed by a subject in a reasonable amount of time. As previously mentioned, increasing the number of items in a measure usually increases the reliability of the measure. As with multiple-choice quizzes, all five types of reliability are applicable to cloze tests. However, unlike multiple-choice quizzes, alternate forms of cloze tests are easy to construct. This means that alternate-form reliability is much more practical to measure.
Cloze tests do not seem to possess many advantages when compared to the common predictive measures $E$ and $v(G)$. $E$ and $v(G)$ do not require the use of human subjects and are easier to construct, administer, and interpret than cloze tests. $E$ and $v(G)$ do have their drawbacks, though. Curtis et al. conclude from their research with these measures:

> Yet, assessing the psychological complexity of software appears to require more than a simple count of operators, operands, and basic control paths. If the ability of complexity metrics to measure programmer performance is to be improved, then metrics must also incorporate measures of phenomena related by psychological principles to the memory, information processing, and problem solving capacities of programmers. [CURT79a, p. 103]

While $E$ and $v(G)$ are independent of many important aspects of software such as commenting, indentation, control flow nesting, etc. and are largely insensitive to psychological phenomena, cloze tests (at least in theory) take into account all aspects of the text material and are sensitive to the psychological characteristics of the comprehending agents. These potential benefits of the cloze test will not be of significance, though, until cloze tests have been shown to be reliable and valid measures within the software domain.

**Previous Research with the Cloze Procedure**

**Prose Domain**

Cloze tests in the prose domain were first systematically studied by Wilson L. Taylor [TAYL53]. Taylor's initial set of small-scale experiments with cloze tests laid the foundation for a great deal of research into the theoretical and methodological issues concerning cloze tests. Hank considers cloze testing to be "one of the most widely researched areas in the field of reading" [HENK81, p. 347]. To date there have been only a few reports that cast doubt on the value of cloze tests within the prose domain [ROB81]. The following is a review of the major issues examined by prose researchers.
Reliability and Validity. Robinson [ROBIBI] concluded from his research review that by the late 1960's the prose community largely regarded cloze testing as a theoretically proven readability measure (even though theoretical experimentation did not stop). As English as a Second Language (ESL) became an increasingly important educational topic in the 1970's, cloze testing rapidly grew in popularity as being a quick and economical language proficiency measure [BROWBO]. Brown [BROWBO] lists over a dozen reliability and validity studies performed between 1970 and 1980 where cloze tests were used in an ESL context. He summarized the results as showing cloze testing to be a generally reliable and valid measure of overall ESL proficiency [BROWBO, pp. 311 and 312].

Deletion Selection. Prose researchers have studied several issues concerning the process of selecting deletions when constructing cloze tests. These issues are: (1) which words should be candidates for deletion, (2) what techniques should be used for selecting the deletions from the candidates, (3) what percentage of words should be deleted, and (4) how many words should be deleted. The most important of these issues is the first. Taylor [TAYL53] strongly argued that, when testing reading comprehensibility, the technique for selecting the cloze deletions should be blind to the properties of the individual words and that all words should be equal candidates for deletion. Words should not be ignored for deletion because they are short, common, or otherwise thought to be too easy to replace if deleted. Likewise, words thought to be unrepresentative or too difficult to replace also should be candidates for deletion. Taylor believed that cloze tests would be biased when passages with different proportions of various "kinds" of words were compared unless the deleted words proportionately represented the kinds of words that occurred in the passages. Taylor also argued for blind selection by presenting many example sentences where "unimportant" or "easy" words seemed more related to sentence comprehension than "important" or "hard" words. The principle of blind selection, though, may seem to lead to tests with questionable validity and reliability. Certainly some words have 100% entropy and are
impossible to replace without guessing. Not only do these words reveal little about the comprehensibility of the text, their unpredictability can be frustrating for subjects, which is a generally undesirable test effect. Also, words that are extremely easy or difficult to replace are poor discriminators and add little to the reliability of the test. Despite these problems, the arguments in favor of blind selection are three-fold: (1) a cloze test should reflect the behaviors of the subjects and not the biases of the researcher, (2) it is largely impossible to a priori judge the "worthiness" of a word for deletion outside the context in which it is used, and (3) the results from improvements on blind selection are not worth the effort.

Taylor [TAYL37] tested the arguments for blind selection by comparing cloze tests where the deletions were made in a blind manner, restricted to only "easy" words (i.e., conjunctions, pronouns, articles, verb auxiliaries), or restricted to "hard" words (i.e., nouns, verbs, adverbs). Taylor found significant correlations between the results of the three types of cloze tests and with tests of pre-reading knowledge (knowledge possessed by subjects before reading the text), immediate text recall ability, and verbal aptitude. The hard-words cloze test correlated the best with the pre-reading knowledge test, but the "blind" cloze test correlated the best with the other two criterion tests.

Building on Taylor's work, Rankin [RANK57, RANK59] hypothesized that by using different deletion selection procedures cloze tests could be designed to measure two different aspects of comprehension. Rankin compared cloze tests constructed based on a blind selection strategy ("structural" cloze tests) to cloze tests where the deletions were only nouns and verbs ("lexical" cloze tests). Rankin found that the structural cloze tests were better than the lexical cloze tests for predicting the results of standardized reading and intelligence tests. Rankin concluded that the structural cloze tests were better indicators of syntactic comprehension and pre-reading knowledge (it is not known by the author if this conflict with Taylor's conclusion has been resolved) while the lexical cloze tests were better measures of post-reading knowledge and "substantive content" comprehension. Greene
[GREE64] also compared a structural cloze test to a lexical cloze test (including adjectives and adverbs, but excluding words not considered predictable by a "superior" reader). Although the lexical cloze test had better reliability (because of greater item homogeneity), better distribution of item difficulties, and better item discrimination, the validity coefficients for the two tests were not significantly different. Greene concluded the extra effort to construct the lexical test was not warranted by the results.

Once it is determined which words may be deleted, one must decide how to select the actual deletions. Given the desire for a cloze test to accurately reflect the contents of the sample text, the theoretically "ideal" technique for deletion selection is totally random selection. A possible alternative is the pseudo-random technique of selecting every nth (e.g., every 5th) word for deletion. Note that with an every-nth deletion strategy there are n possible deletion patterns (depending on the starting word). A drawback of an every-nth deletion technique is that it may get in sync with an author's writing style and not accurately reflect the makeup of the text. Several researchers have noted that different every-nth deletion patterns can differentially affect performance [see BOYCB2, ENT178, MERE78, PORT78, BORM64b]. After comparing different every-nth deletion patterns to different totally random patterns, Meredith and Vaughan [MERE78] concluded that totally random patterns provide a more stable measure of text comprehension. A potential drawback of a totally random deletion technique, though, is that in places the deletions may fall too close together and not provide sufficient context for the determination of the missing words. However, most researchers trust that the two methods of every-nth and totally random deletion approximate each other in the long run, and see the every-nth technique as having the additional advantages of being easier to implement and always providing the maximum amount of context for each deletion. Bormuth [BORM64b] suggests that several every-nth cloze tests be used to insure dependable results.
Having selected a deletion technique, one must decide what percentage of words to delete. Taylor [TAYL53] suggested that from ten to twenty percent of the words in a text be deleted. Aborn et al. [ABOR59] showed that increasing the context for deletions by having a deletion percentage below ten percent did not increase the predictability of the deleted words and MacGinitie [MACG61] showed that decreasing the context with a deletion rate above twenty percent tended to make the prediction of the deleted words interdependent. A deletion rate from ten to twenty percent is, therefore, considered optimal.

The last issue is concerned with the total number of deletions to be made or, alternatively, how long the measured passage must be. Bormuth [BORM64a] found that a total of 50 deletions is optimal. Fewer deletions decrease the sensitivity of the measure and more deletions are unnecessary for accurate measurement.

**Blank Format.** There is some debate over the matter of what form the blank lines in a cloze test should have. The main concern is whether the blank lines should provide cues for the missing word such as word-length cues or initial-letter cues [see RUSH70]. Anderson [ANDE70] found that the common restriction that all blanks be of a uniform length was unnecessary and suggested that cloze tests be simply made from photocopies of the original text. This process would make cloze tests much easier to construct than the usual method of retyping the original text. On the other hand, Culhane [CULH70] found that cued-length blanks adversely affected the usefulness of cloze tests and supported uniform-length blanks. Most researchers take the more conservative approach and use blanks with a uniform length of ten to fifteen spaces.

**Scoring.** The manner in which a test is scored can determine the theoretical soundness and usefulness of the test. The same test may prove to be invalid when scored one way and valid when scored another way. Brown [BROW80] has examined the four most prominent cloze test scoring methods: exact-answer, acceptable-answer, clozentropy, and multiple-choice.
The exact-answer (also known as "verbatim") and acceptable-answer (or "synonymic") scoring techniques have been used since Taylor's original study [TAYL53]. A verbatim scoring method accepts as correct only responses that exactly match the missing word from the original text (with the possible exception of obvious spelling errors). A synonymic scoring strategy allows any response that preserves the meaning of the original text or is otherwise considered contextually acceptable. Studies have shown that for native speakers there is no significant statistical difference between these two methods [see HENK81, MCKE76, BORM65, RUDD64, TAYL53] even though the verbatim technique tends to deflate scores and has less face validity. Oller [OLLE72] has concluded, though, that the synonymic scoring method is better than verbatim when testing subjects having English as a second language. Because of the subjective nature and greater scoring difficulties with synonymic scoring, most researchers choose the verbatim scoring technique when working with native speakers.

The clozentropy and multiple-choice scoring methods have only recently been used by researchers. The clozentropy technique attempts to improve on the synonymic scoring method by using information-theoretic techniques to calculate the degree to which a subject's responses differ from those of the group with which the subject is being compared [see REIL71, DARN70]. With the multiple-choice scoring technique, subjects choose their response from alternatives presented for each blank [see JONZ76]. Multiple-choice scoring differs from the other three methods in that multiple-choice is probably more of a test of a subject's ability to recognize a correct response while the other methods are more of a test of a subject's ability to generate a correct response.

Brown [BROW80] conducted an experiment in which he compared all four scoring techniques on the bases of standard error of measurement (measurement of the range in which test scores should fall if the test is taken repeatedly), reliability and validity tests, mean item facility (measurement of the difficulty of a test), mean item discrimination (measurement of a test's ability to discriminate between high and low
performing subjects), and ease of use. Brown's study showed that all the methods were reliable and valid and highly intercorrelated. Brown concluded that the synonymic technique, which was the most reliable and had the best mean item facility and mean item discrimination was the overall best measure with the verbatim technique an extremely close second. A disadvantage of cloze tests is that no matter the scoring method, scores are in general low (around 50 percent). Such low scores can obscure ceiling effects and frustrate high ability subjects [ROB181].

The Standard Cloze Procedure. A standard cloze test is formed by following a standard "cloze procedure", that is, a set of rules by which a cloze test is constructed. The rules and terminology that Borauth [BORM62] set forth are now almost totally accepted by all prose researchers [ROB181] and form the basis of the standard cloze procedure for prose analysis. A standard cloze test for prose material obeys the following guidelines [ROB181]:

1. Words are defined by the "white space" the author of the text uses to delimit the words. All punctuation is treated as white space except apostrophes and sometimes hyphens. A hyphen is considered white space only when each part of the hyphenated word is a proper word (i.e., not a prefix or other word stem).

2. Blanks are a uniform length (generally ten to fifteen spaces) so as not to cue the missing words.

3. Every 5th word is deleted beginning with a word randomly selected from the first five candidate words. Fifty-word lead-in and ending sections without blanks can be used to allow subjects some familiarity with the subject matter and style of the text.

4. There should be at least 50 deletions on a cloze test. Taylor [TAYL53] also argued that when using cloze tests to compare passages that the passages should be of approximately equal length and contain the same number of deletions.
(5) Verbatim scoring allowing for obvious misspellings should be used.

Other cloze procedures have been used by prose researchers with varying levels of success, but most cloze tests follow the above methodology. In a recent study, Henk [HENKB1] investigated the interaction of deletion pattern (every-5th versus totally random), blank format (uniform length versus length-cued), and scoring mode (verbatim versus synonymic). Henk's research supported the notion of little difference between verbatim and synonymic scoring and also showed a random/cued/verbatim cloze procedure as being a "theoretically and empirically promising alternative" to the standard every-5th/uniform/verbatim procedure [HENKB1, p. 355]. Henk theorized that the drawback of too close deletions usually associated with random deletion was circumvented by the presence of cued blanks and concluded that the totally random aspect of the random/cued/verbatim procedure made the procedure more objective and generalizable than the common every-5th/uniform/verbatim procedure.

Software Domain

Only in the last few years have software engineers begun to take interest in the cloze procedure. While not specifically investigating the cloze procedure themselves, Shneiderman [SHNE80] and Kolton et al. [KOLT83] have suggested that the cloze procedure might be adapted to the purposes of the software engineer. Beyond this, the author is aware of only four software engineering research efforts that have incorporated cloze tests in some manner. Two of the investigations used cloze procedures as a research tool incidental to the main research goals and the other two were oriented toward uncovering the properties of cloze tests in the software domain.

Ehrlich and Soloway [EHRL82] conducted a research effort that used cloze tests as a research tool. Ehrlich and Soloway examined the hypothesis that experienced programmers possess more preconceived notions ("tacit knowledge") about the construction and inner-workings of software than novice programmers. They confirmed their hypothesis by showing that experienced programmers were faster, more accurate, and less variable
than novice programmers when filling-in deleted lines in programs. An examination of subject responses to blank program lines also revealed that experienced programmers reacted with greater accuracy, less variability, but less speed than novices when programs were designed to conflict with accepted programming conventions. The behavior of the novices was about the same as with the conventional programs. This was interpreted as indicating that the experienced subjects were confused (slowed down) by programs that violated the accepted programming conventions while the novice programmers were oblivious to the violation of convention.

A. F. Norcio is another researcher to have used cloze testing as a research tool. Norcio conducted a series of experiments using cloze tests to demonstrate that indentation and commenting help program comprehensibility [NORC82a, NORC82b, NORC81a, NORC81b]. For the indentation experiments Norcio constructed two identical implementations of an algorithm, one using indentation and the other not. Certain lines were then deleted from the two programs. It was shown that the lines in the program with indentation were more likely to be filled-in correctly than the corresponding line in the unindented program. Similar experiments also successfully demonstrated the benefit of comments.

Unlike the above researchers, who simply borrowed cloze testing as a research tool from the prose domain, C. R. Cook investigated the utility of cloze testing as a software comprehension measure [COOK82]. Also unlike the above researchers, Cook used a cloze procedure similar to the standard cloze procedure in the prose domain. For his experiment Cook developed two implementations of an algorithm. By using a short-answer/multiple-choice comprehension quiz, one implementation was demonstrated to be more difficult to comprehend than the other. Cook constructed cloze tests from the two implementations by deleting every fifth token from the programs (ignoring a few punctuation items). Cook found that scores on the cloze tests closely paralleled scores on the corresponding comprehension quiz.
Entin, who has experimented with the cloze procedure in the prose domain, also conducted an experiment to assess the feasibility of using a cloze procedure to measure program comprehension [ENTI84]. Like Cook, Entin found cloze test scores to be positively related to other measures of program comprehension. Entin also found that a verbatim scoring technique was consistent with a more costly synonymic scoring technique.

The Need for More Research in the Software Domain

Because of the qualitative differences between software and prose, there is little reason to suspect that the standard cloze procedure for the prose domain will function equally well in the software domain without at least some modification. Software is a much more highly structured communications medium than prose, both syntactically and semantically. Unlike prose, software possesses a rigid inner logic which largely defines and fixes the need, purpose, and location of each token. Also, programming languages have small vocabularies and few grammar rules that are usually completely mastered by proficient users of the languages. This is in contrast to the large, ever-changing vocabularies and largely uncharted grammars of natural languages that sustain many students. The inherent differences between software and prose require that cloze testing be investigated in the software domain as well as it has been tested in the prose domain. Shortcomings in the research that has been performed to date and the many as yet unanswered questions concerning software cloze testing suggest there is still a need for more research on the theoretical and methodological issues concerning software cloze testing.

Shortcomings of Previous Research

While the above experimental results give support to the notion that cloze testing can be a valuable tool in the software domain, the experiments suffer shortcomings that should be overcome in any future research. A major shortcoming is that only Cook and Entin used more accepted measures to validate their cloze results. The unknown nature of cloze testing in the software domain makes software cloze results untrustworthy and requires that they be corroborated with already
trusted procedures. Demonstrating that cloze results agree with the results from commonly accepted measures helps to establish the validity of cloze testing.

The second shortcoming of the previous software cloze research is that all the experiments made use of only "toy" programs that were approximately 30 lines in length. If cloze procedures are to be of use in the "real world", then their suitability must be demonstrated by using programs with characteristics similar to software encountered in that domain.

A shortcoming of the experiments as a whole, instead of individually, is that they are all too dissimilar to give convergent support to research issues or to allow useful comparisons. For example, although both Cook and Entin used cloze tests with every fifth token deleted, their definitions of what constituted a token differed. Future research efforts should attempt to validate and build on previous efforts in such a manner that a body of sound knowledge concerning software cloze testing may be accumulated.

Unanswered Questions

A cloze procedure is defined by several parameters (e.g., scoring technique, deletion technique, etc.) and there are many questions that have yet to be examined, let alone answered, concerning these parameters for cloze testing in the software domain. The most important questions are concerned with whether cloze tests are a reliable and valid measure of software comprehensibility. Entin states that "Various deletion systems need to be tried to see which one yields the most reliable and valid results." [ENTI94, p. 46]. By comparing their cloze results to those of more accepted measures, Cook and Entin provide some positive evidence of the validity of software cloze tests, but there have been no studies that have examined the reliability of software cloze tests (even though Cook's and Entin's results are open to internal consistency testing). Since all four of the above previous research efforts followed different (and sometimes unspecified) cloze procedures, there
is little consensus on what program tokens should be candidates for deletion, what techniques should be used to select the deleted tokens, or what methods are best for scoring. This research is intended to shed some light on these and other topics.
Validating the Experimental Materials

A main goal of this research was to examine the validity of cloze testing as a software comprehensibility measure. The establishment of the validity of a test generally requires independent, external criteria of whatever function the test is designed to measure. A common method for exploring the validity of a new metric is to apply the new metric and another, more standard and accepted metric, to the same test materials and compare the results. If the results from the new metric are similar to the results from the accepted metric, then support is given to the validity of the new metric. If this process is performed repeatedly using various test materials with comparable results, then the validity of the new metric is eventually established.

An ideal experimental situation for testing a new metric is one where there already exists standardized test materials with which to work. In such a situation all that needs to be done to test a new metric is to apply the metric to the standardized materials and compare the new results to the already established results. Although such previously validated test materials often exist for prose studies, this is not true for software research. Before cloze testing could be experimentally examined in this research effort, it was first necessary to develop appropriate test materials and to establish their comprehensibility using an accepted technique.

The objective of the pilot study was to develop two versions of a single program and to determine the comprehensibility of the two programs using a multiple-choice comprehension quiz. A multiple-choice quiz format was selected as the “standard” measure of comprehensibility because of its wide acceptance within the software engineering research community, its familiarity by subjects, and its ease of application. The decision to
develop two programs instead of one was based on the nature of the cloze test and multiple-choice quiz measures. In general, an individual comprehension quiz score is meaningless. That is, a comprehensibility score of 50% may either represent a low, medium, or high degree of comprehensibility, depending on characteristics of the subject using the measure, the test materials measured, and the measure itself. This problem can be overcome by measuring two sets of test materials. Although the meanings of the individual program scores are still unclear, the relative ordering of the scores is meaningful. It is commonly assumed that both multiple-choice comprehension quiz scores and cloze test scores possess the properties of an ordinal scale measure and that it is therefore meaningful to rank order the scores. Thus, if a measure assigns a certain comprehensibility score to one program and a significantly higher comprehensibility score to another program, then the first program can be considered less comprehensible than the second.

Usually software engineers do not just want to know that two programs differ in comprehensibility, they also want to know the causes for the difference. To discover the software factors that affect comprehensibility, software engineers commonly experiment with programs that differ only in one characteristic that is believed to influence program comprehensibility (such as control flow structures). If comprehensibility differences are found, then it is reasonable to assume that they are due to the one program characteristic in which the programs differ. Although it was not a goal of this research to determine which software characteristics can affect comprehensibility, common experimental design principles were followed and comprehensibility comparisons were made using programs that were similar across most dimensions. In this manner confounding effects are minimized and the process of interpreting experimental results is facilitated. The programs will be described in more detail in the "Materials" section of Pilot Study I.

There are two basic experimental designs one may use when performing human subject experimentation such as a comprehension study. One is a "within-subjects" design and the other is a "between-subjects" design.
A within-subjects design, such as has been used in Entin's [ENT184] cloze research, "measures" each subject in a group more than once and then compares the results of the multiple measures. A between-subjects experiment such as Cook's [COOK82] measures each subject in a group only once, but different groups are measured using various metrics and the responses of the multiple groups are then compared. An advantage of a within-subjects experimental design over a between-subjects design is that variability due to subject differences, which can help hide experimental effects, is minimized. Disadvantages of a within-subjects design are that it can be difficult to measure a subject repeatedly in the often limited amount of time available for an experiment and that there can be carryover effects from one condition to the next. A goal of this research effort was to have subjects work with less toy-like programs than had been used in previous software cloze research. Since time constraints would allow each subject to work with only one program, a between-subjects experimental design was selected for this study.

Pilot Study 1

Design and Overview. The purpose of Pilot Study 1 was to establish the relative comprehensibility of two versions of a program using a standard comprehension measure. In subsequent experiments the relative comprehensibility of the same two program versions was measured using cloze tests and the results of the two testing procedures were compared.

A single-factor experiment was performed with the independent variable being program version ("easy", "hard") and the two dependent variables being the score on a multiple-choice comprehension quiz and the time to complete the quiz. The hypotheses of the experiment were: (1) the scores on the multiple-choice quiz for the "easy" program would be higher than scores on the quiz for the "hard" program and (2) subjects with the "easy" program would complete the multiple-choice quiz in a shorter time than subjects with the "hard" program.

Subjects. Eighteen students, including graduates and undergraduates, in an upper-level software engineering course at The Ohio State University participated in the experiment as partial fulfillment of the course
Materials. Experimental materials included a background survey [Appendix A], a general instruction sheet [Appendix B], a program booklet for each of the two programs containing a program description sheet [Appendix C] and a listing of one of the programs [Appendices D and E], quiz booklets containing a quiz instruction sheet [Appendix F] and a multiple-choice comprehension quiz tailored to one of the programs [Appendices G and H], and a post-experimental questionnaire [Appendix I].

The background survey consisted of questions designed to gather biographical information about the subjects and to assess their programming experience and aptitude. The general instruction sheet informed the subjects of the approximate amount of time allotted for the experiment, that they should work as quickly and accurately as possible, that not all subjects were working with the same materials and thus some subjects could be expected to finish before others, and that they were to follow the instructions in the quiz booklet when told the experiment had begun. The program description sheet defined the function of the corresponding program, described the error processing performed by the program, and discussed the structure of the program. The program description sheet was identical for both program implementations. The comprehension quiz instruction sheet told the subjects to indicate the best alternative answer for each of the multiple-choice questions and again reminded the subjects to raise their hand any time they had a question. The multiple-choice quizzes consisted of fifteen questions, each with five alternative responses. The two quizzes were conceptually identical, but were tailored to the corresponding program and by necessity differed in minor details such as program line number references. The multiple-choice questions were of two general types: tracing questions and "what if" questions. The tracing questions defined a "start" state for the program (by specifying input data, or the value of certain variables, etc.) and then asked the subject about the status of some aspect of the program at a given "stop" state. The
"what-if" questions asked the subject what would be the effect of various circumstances such as a specific change to the program or a given error in the input data. The post-experimental questionnaire contained four questions designed to assess a subject's perception of (1) their familiarity with the program worked with in the experiment, (2) how well the program was comprehended, (3) how easy the program was to comprehend, and (4) how well written the program was considered to be. The post-experimental questionnaire also asked the subjects for any comments they might have concerning the experiment.

The two programs used in the experiment were Fortran adaptations of published COBOL programs [DHYEB1, LEVYB2]. A poll taken of the subjects before the study indicated that Fortran was the language with which the subjects were the most familiar. Transcriptions of the two programs are in appendices D and E.

The two program versions were functionally equivalent in the sense that the domains of proper input for the two programs were identical and that identical proper input for each program would produce identical output. Identical, but improper input to the two programs could cause non-identical output. Both of the programs solved the familiar master file update problem using the "Balanced-Line Algorithm" (see DIJK76 for a description of the algorithm). The programs were similar along the following dimensions: types of operators and operands, spacing and indentation, commenting, variable naming conventions, and length (approximately 200 lines). Also, only one-in-one-out ("structured") control structures were used. The main difference between the two programs was that one program was a "conventional" implementation of the Balanced-Line Algorithm while the other program was additionally structured by applying principles of abstract data types such as information hiding and encapsulation. The "encapsulated" version of the program consisted of two large modules (or "capsules") and two "driver" programs. One capsule performed all the input-oriented processing and the other capsule performed all the output-oriented processing. It was hypothesized that the encapsulated version of the program would be more difficult to comprehend than the conventional version. This hypothesis
is counter to the standard wisdom that the use of abstract data types improves program quality. The reasoning behind the hypothesis was that the encapsulated program contained more implementation "overhead" than the conventional version of the program. This overhead is believed due largely to the need for the capsules to share information with each other. In fact, Meeson and Pyster [MEES71] found that the use of abstract data types decreased program modifiability because of a "bloating" of program size. This "bloating" may also decrease program comprehensibility. It may be that data abstraction does indeed improve program quality, but only in cases where the "good" effects of data abstraction outweigh its "bad" effects. Programs may have to be large or complex before the scales tip in favor of data abstraction. It may also be the case that data abstraction is only effective for certain types of programs or when using certain programming languages such as CLU. The Master File Update problem may not be a proper application for data abstraction and languages such as Fortran, COBOL, and PL/I may not be proper languages for illustrating the advantages of abstract data types. (Henceforth the conventional and encapsulated versions of the program will also be called "easy" and "hard", respectively.)

Procedure. The subjects were run in one group in a classroom setting. Subjects were balanced across treatment conditions based on the number of Fortran programs they had written as reported in a poll taken in class a few days before the experiment. Each subject was given a booklet containing all the experimental materials except the post-experimental questionnaire. Half of the subjects received one version of the master file update program and half received the other version. Subjects were instructed to not open their experimental booklet until so signaled. The timing of the experiment began when the subjects were told to open and read the instructions inside the experimental booklet. After completion of the comprehension quiz, a subject would returned the booklet to the experimenter, the time of completion for the subject was noted, and the subject was given the post-experimental questionnaire. The subjects were allowed to leave after completing the post-experimental questionnaire. Time was called after 45 minutes and the
remaining subjects turned in their experimental booklets and completed the post-experimental questionnaire. There were five late arrivals who started the experiment up to fifteen minutes after the rest of the group. These subjects were instructed that they would be informed individually about when time had run out for the experiment and were allowed the same 45 minutes as the other subjects. All five of the late subjects were in the "easy" treatment condition.

Results. Although it is not common practice to examine the reliability of a measure planned to have a short life-span, the reliability of the two multiple-choice quizzes was calculated. The reliability estimates for the tests provide an indication of the validity of the results and also provide a basis for comparison with the reliability estimates for the cloze tests used in future experiments. As indicated in Chapter I, the reliability of the quizzes was estimated using Kuder-Richardson formula number eight (KR-8). The results of the reliability analysis are shown in Table 3.

<table>
<thead>
<tr>
<th>Quiz (Program)</th>
<th>Kuder-Richardson Reliability Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>.57</td>
</tr>
<tr>
<td>Hard</td>
<td>.51</td>
</tr>
</tbody>
</table>

Although the inter-item consistency of each of the two quizzes is low, the reliability estimates for both the easy and hard program quizzes are considered acceptable since they are above the previously stated .50 lower bound.

The time and score experimental results were examined using one-tailed t-tests. As revealed in Table 4, subjects with the easy version of the program had significantly higher scores than subjects with the hard version of the program. This result supported the hypothesis that the conventional version of the program was easier to comprehend than the
hard version. On the other hand, although the subjects with the "easy" program spent less time on the average working on the comprehension quiz than the subjects with the "hard" program, there was no significant difference in the timing results for the two versions. These data did not support the experimental hypothesis, but inspection of the timing information suggests that a ceiling effect may have occurred since a majority of the subjects did not turn in their test materials before time was called. Although the subjects may have desired more time to work with the test materials, the subjects did supply answers to a majority of the questions. Subjects working with the easy program left an average of 2 questions unanswered. Subjects working with the hard program left an average of 3 questions unanswered.

Table 4
Pilot Study 1 t-test Analysis of Time and Score

<table>
<thead>
<tr>
<th>Program Type*</th>
<th>Conventional</th>
<th>Encapsulated</th>
<th>t</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>M 42.4 min</td>
<td>44.2 min</td>
<td>-1.36</td>
<td>.10</td>
</tr>
<tr>
<td></td>
<td>SD 3.2</td>
<td>2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Score</td>
<td>M 5.0</td>
<td>2.8</td>
<td>2.63</td>
<td>.01</td>
</tr>
<tr>
<td></td>
<td>SD 2.0</td>
<td>1.6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*a n = 9 for each group.

*Maximum possible time = 45 min., maximum possible score = 15 points.

Responses to the background survey and post-experimental questionnaire were also analyzed. The means and standard deviations of the responses are reported in Appendix J and correlations of the responses with the independent and dependent variables are reported in Appendix K. The background survey and the post-experimental questionnaire were mainly used as a tool to describe the subjects used in the experiments. The reported statistics should help other researchers in comparing their experiments to the author's experiments.

Because the post-experimental questions were all highly subjective in nature, only "dramatic" differences between the treatment groups will be
considered important (of course, this too is subjective). The major benefit of the post-experimental questionnaire is that it asked the subjects for any comments they might have concerning the experiment. If any unusual situations were to arise during the experiment or if any subjects were to react to the experiment in an unusual manner, it was hoped that the comments would alert the experimenter to the event.

Conclusions concerning relationships between the background survey and post-experimental questionnaire variables and the dependent variables will, in general, be avoided for the individual experiments performed in this research. Performing many correlations for each experiment increases the likelihood that spurious results may occur. Also, the small number of largely homogeneous subjects used in each experiment can easily lead to a situation where a statistically significant effect is caused by only one or two subjects. It can also be difficult to correctly interpret the meaning or importance of some of the correlations. For example, class rank was negatively correlated with score ($r = -0.85, p < 0.01$) for the subjects working with the easy program. This seems counter-intuitive since one might expect higher ranking subjects to perform better. This result may be explained by the fact that all the subjects were either seniors (class rank = 4) or graduate students (class rank = 5) and that the lower ranking subjects had taken a larger number of programming classes than the higher ranking subjects (an average of 10 classes for the seniors compared to an average of 6 classes for the graduate students ($r = 0.67, p < 0.06$)). One might expect this result since senior computer science student are usually in the process of finishing a degree in computer science while graduate students may just be starting a degree. To avoid spurious results, only correlations within an individual experiment that hold for each treatment group as well as the combined groups will be mentioned. (Although there were nine subjects in each of the treatment groups of the current experiment, a subject in the hard group did not properly complete the background survey and the post-experimental questionnaire and was dropped from the analysis of these variables.)
Because of the small number of subjects, the large differences in number of Fortran programs written by individual subjects, and the differences in subject mortality for the treatment groups, the mean (and median) number of Fortran programs written by each treatment group were not similar even though the treatment groups were "balanced" on this criterion (a mean and median of 10 and 20.4, respectively, for the easy group and 35 and 51.5, respectively, for the hard group). However, the correlational analysis of the background survey variables and the independent variable showed that there was no significant correlation between the number of Fortran programs written by a subject and the treatment group in which the subject was placed. This is an indication that subjects were properly balanced across treatment conditions according to the balancing criterion. There was no background survey or post-experimental questionnaire variable that was significantly correlated for both the individual treatment groups and the combined group.

Discussion. An easy (conventional) version and a hard (encapsulated) version of a master file update program and two corresponding multiple-choice comprehension quizzes were developed in Pilot Study 1. Subject scores on the multiple-choice quizzes supported the hypothesis that the conventional version of the program was easier to comprehend than the encapsulated version of the program. Subjects did not differ in their times to complete the quizzes, but this failure was believed to be due to a ceiling effect. The reliability of the two quizzes was found to be acceptable.

It was felt that stronger experimental results were necessary to more fully support the hypothesis concerning the comprehensibility of the two versions of the program. A second pilot study was conducted where the subjects were allowed more time than in the first pilot study to complete the quizzes. It was believed that the additional amount of time would alleviate the ceiling effect on time scores found in the first pilot study and that the time scores would then also support the experimental hypothesis. In addition it was hoped that the extra time would allow subjects to complete more items and thus improve the
reliability of the quizzes.

**Pilot Study 2**

Pilot Study 2 was a replication of Pilot Study 1 with three modifications: (1) more time was allowed for subjects to complete their comprehension quiz, (2) the programs were coded in PL/I instead of Fortran, and (3) the subjects were volunteers hired to participate in the study instead of students fulfilling a course requirement. The test time was increased in the second study in an attempt to overcome a possible ceiling effect for time scores found in the first pilot study and to improve the reliability of the quizzes. The programming language change was made because, unlike the subjects in Pilot Study 1, the subjects for Pilot Study 2 indicated in a biographical survey completed a few days before the study that PL/I was the language with which they were most familiar. The increase in the time allowed for the study made it necessary to hire subjects outside a classroom setting since more than a class period was needed for the study.

The purpose of Pilot Study 2 was to obtain additional evidence that the conventional implementation of the master file update program from Pilot Study 1 was easier to comprehend than the encapsulated implementation. It was hoped that the hypothesis would be supported by both the timing information and scores on the comprehension quizzes instead of just the scores alone as was the case in Pilot Study 1. It was also hoped that better reliability than was determined in the first pilot study could be obtained for the quizzes.

**Design and Overview.** As in Pilot Study 1, a single-factor experiment was performed with the independent variable being program version ("easy" or conventional, and "hard" or encapsulated) and the dependent variables being the score on a multiple-choice comprehension quiz and the time to complete the quiz. The hypotheses of the experiment remained the same as in Pilot Study 1: that subject scores on the multiple-choice quiz for the easy program would be higher than subject scores on the quiz for the hard program and that the time for subjects
to complete the quiz for the easy program would be less than the time for subjects to complete the quiz for the hard program.

Subjects. Eighteen students currently taking at least one of four different advanced level computer science courses at The Ohio State University participated in the experiment. All but one subject were undergraduates. All the subjects were paid volunteers solicited via announcements read in classrooms and posted on bulletin boards. Each subject was paid $5.00 per hour for a maximum of two hours of participation in the experiment. A signed experimentation consent form was obtained from each subject before the start of the experiment [see Appendix L].

Materials. The experimental materials used in the second pilot study were conceptually the same as the materials used in the first pilot study. The general instruction sheet for the experiment was identical to the one used in the first pilot study except that the subjects were told that they had approximately an hour and 45 minutes to complete the experiment instead of 45 minutes. As an aid to the less advanced subjects of Pilot Study 2, the program description used in the first pilot study was augmented for Pilot Study 2 to include a short description of ENTRY statements and STATIC variables, two of the more difficult constructs used in the implementation of the PL/I programs [see Appendix M for the augmented program description]. The change in style used to declare variables in PL/I caused the length of the program to increase from about 200 lines for each program version in Fortran to approximately 310 and 250 lines for the easy and hard versions, respectively, in PL/I. See Appendix N and Appendix O for transcriptions of the two PL/I programs. Changes in the implementation language also required that minor details such as variable names and program line references be changed in the quizzes for the two programs [Appendices P and Q].

Procedure. The subjects were run at the convenience of their schedule in small groups and individually. All subjects were run within a week of each other. The procedure for the experiment was the same as in
Pilot Study 1 except that (1) an hour and 45 minutes was allowed for the experiment instead of just 45 minutes, (2) subjects were balanced across treatment conditions based on the number of PI/I programs they had written as reported in a background survey completed before the experiment, and (3) subjects were paid for their participation at the end of the experiment.

Results. All the subjects completed the experiment in the allotted time. As in Pilot Study 1, the reliability of the two quizzes was analyzed using the Kuder-Richardson reliability estimate. The results of the analysis are reported in Table 5.

### Table 5
Pilot Study 2 Reliability Analysis

<table>
<thead>
<tr>
<th>Quiz (Program)</th>
<th>Kuder-Richardson Reliability Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>.32</td>
</tr>
<tr>
<td>Hard</td>
<td>.75</td>
</tr>
</tbody>
</table>

The reliability estimate of .75 for the quiz over the hard version of the program was satisfactory, but the reliability estimate for the easy program quiz indicated that there was little inter-item consistency. An item analysis of the easy program quiz revealed that four of the fifteen quiz items were negatively correlated with total quiz scores (specifically, items number 8, 12, 13, and 15). This meant that subjects that performed well on the quiz as a whole tended to perform poorly on these individual items and vice-versa. However, the negative correlations were low (-.26 was the "worst") and none were significant. In general, though, items on a comprehension quiz that correlate negatively with overall score are considered poor measures of comprehension since they violate the assumption that better comprehension is indicated by better performance on the individual quiz items. This also violates an assumption of the Kuder-Richardson reliability estimate. An analysis was performed to see if deleting some of these test items would leave a reliable subset of items. A
conservative approach was taken and as few items were deleted as possible since decreasing the number of items in a measure increases the difficulty in achieving significant results. As expected, deleting each negatively correlated item increased the Kuder-Richardson reliability estimate. Table 6 summarizes the results from a Kuder-Richardson analysis after discarding from both quizzes the two items of the easy program quiz that had the largest negative correlation with total quiz score (items 8 and 13). Note that only one of the four negatively correlated items from the easy program quiz (item 15) also correlated negatively with the overall score on the hard program quiz. Further, in the first pilot study none of the questions correlated negatively with overall score for the easy program quiz and only one question (item 13) correlated negatively with overall score for the hard program quiz. These inconsistent results and the low, non-significant negative correlations are interpreted as indications that the four negatively correlated items were not poor test items in general.

Table 6
Pilot Study 2 Reliability Analysis After Deletion from Conventional Program Quiz of Two Items Negatively Correlated with Overall Score

<table>
<thead>
<tr>
<th>Quiz (Program)</th>
<th>Kuder-Richardson Reliability Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>.53</td>
</tr>
<tr>
<td>Hard</td>
<td>.71</td>
</tr>
</tbody>
</table>

Inspection of Table 6 reveals that after the deletion of the two questions from the quizzes the reliability of the easy program quiz increased to an acceptable value (from .32 to .53) and the reliability of the hard program quiz did not change appreciably (from .75 to .71).

Table 7 shows the analysis of the time and score results with the entire set of questions and Table 8 shows the analysis without the two items that were deleted to improve the reliability of the easy program quiz. With or without the two possibly poor test items, one-tailed t-tests
showed the same significant score effect as found in Pilot Study 1, but this time there was also a significant time effect. Subjects working with the easy version of the program had significantly higher scores and took significantly less time to complete the multiple-choice comprehension quiz than did the subjects working with the hard version of the program. Except for one subject, all subjects at least attempted all the questions. One subject working with the easy program quiz left one item unanswered.

Table 7
Pilot Study 2 t-test Analysis of Time and Score

<table>
<thead>
<tr>
<th>Dependent Variable</th>
<th>Program Type</th>
<th>Conventional</th>
<th>Encapsulated</th>
<th>t</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>M 64.3 min</td>
<td>79.1 min</td>
<td>-2.18</td>
<td>.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SD 11.2</td>
<td>17.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Score</td>
<td>M 6.9</td>
<td>4.8</td>
<td>2.02</td>
<td>.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SD 1.6</td>
<td>2.7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*n = 9 for each group.

Maximum possible time = 1 hr. 45 min., maximum possible score = 15 points.

Table 8
Pilot Study 2 t-test Analysis of Score After Deletion from Conventional Program Quiz of Two Items Negatively Correlated with Overall Score

<table>
<thead>
<tr>
<th>Dependent Variable</th>
<th>Program Type</th>
<th>Conventional</th>
<th>Encapsulated</th>
<th>t</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>M 6.1</td>
<td>4.0</td>
<td>2.10</td>
<td>.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SD 2.0</td>
<td>2.3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*n = 9 for each group.

Maximum possible score = 13 points, time scores not re-analyzed since they were not affected by deletion.

Responses to the background survey and post-experimental questionnaire were also analyzed. The means and standard deviations of the responses are reported in Appendix R and correlations of the responses with the
independent and dependent variables are reported in Appendix S. As previously stated, because of the chances of spurious correlations, only significant correlations that hold for each treatment group as well as the groups combined for an individual experiment will be reported.

The correlational analysis of the background survey variables and the independent variable showed that there was no significant correlation between the number of PL/I programs written by a subject and the treatment group in which the subject was placed. The mean number of PL/I programs written for each group were also similar (26.3 for the easy group and 27.8 for the hard group). These results indicate that subjects were properly balanced across treatment conditions according to the balancing criterion. Further, the number of years that a subject had programmed and the number of years a subject had programmed with PL/I were both positively correlated with comprehension quiz score for the easy and hard treatment groups and the groups combined. That is, the more years a subject had programmed and the more years a subject had programmed specifically with PL/I, the better the subject scored on the comprehension quiz. This result is intuitively satisfying and supports the notion of balancing subjects across treatment conditions based on the implementation language experience dimension. There were no other background survey or post-experimental questionnaire variables that were significantly correlated for both the individual treatment groups and the combined group.

Discussion. Pilot Study 2 was a conceptual replication of Pilot Study 1 using a different subject pool, different implementation language, and different time limits. The hypothesis that the conventional implementation of the master file update program is easier to comprehend than the encapsulated implementation was again supported by the comprehension test score data. The supposed time ceiling effect of Pilot Study 1 was successfully alleviated in Pilot Study 2 by increasing the test taking time from 45 minutes to an hour and 45 minutes and the time data now also supported the experimental hypothesis. The reliability of the comprehension measures was shown to be acceptable, though only after two comprehension questions that negatively correlated
with overall test scores were discarded from the analysis.

It would appear from these results that the relative comprehensibility of the two program versions has been established and that it is appropriate to investigate whether cloze tests can also be used to demonstrate the same relative comprehensibility.
Chapter III detailed how a multiple-choice comprehension quiz, a standard measure of software comprehensibility, was used to determine the relative comprehensibility of two versions of a master file update program. It has been previously noted that cloze tests exhibit several advantages over multiple-choice tests as a comprehension measure within the prose domain (namely, cloze tests may be standardized, are more independent of materials extraneous to the text being measured, have a greater ease of construction, and are more reliable). It has also been noted, though, that more research is needed before cloze tests may be used within the software domain with the same confidence with which multiple-choice tests are used. The purpose of the experiments presented in this chapter was to determine if cloze tests are a reliable and valid measure of software comprehensibility and whether the benefits of cloze tests manifested in the prose domain are also manifested in the software domain. In the first cloze experiment, cloze tests were used to measure the comprehensibility of the same two programs previously measured with the multiple-choice quiz in the pilot studies.

Agreement of the results of the cloze test experiment with the results of the multiple-choice quiz studies would give support to the validity of cloze tests as a comprehensibility measure within the software domain.

Both Cook (COOK82) and Entin (ENTI84) constructed their cloze tests based on the standard cloze procedure for the prose domain (see Chapter II for a description of the standard prose cloze procedure). Due to the positive results obtained by Cook and Entin and the reliable and valid nature of the standard cloze procedure in the prose domain, the cloze tests used in the current experiments were also constructed using the
standard prose cloze procedure.

The major difference between the standard cloze procedure of the prose domain and the cloze procedures used by Cook and Entin is in the definition of a "word" (an item eligible for deletion in the construction of a cloze test). As described in Chapter II, a word in the prose domain is defined by the "white space" the author of the text used to delimit the words. Except for apostrophes and sometimes hyphens, all punctuation is treated as white space in the prose domain. The comparable notion for a prose word in the software domain is that of a "token" as defined by a software lexical analyzer. A deficiency in considering tokens comparable to words is that lexical analyzers generally consider literal strings and comments as single tokens even though they may contain many conceptually independent symbols. Entin bypassed this intuitively unappealing situation by using the prose definition of a word when considering the items within comments and literals. Cook did not address the issue of how to determine what constituted a word when considering comments and literals (his programs contained neither). Cook initially considered all tokens as words, except that (Pascal) punctuation tokens (semicolon, colon, and comma) were treated as white space and ignored. Cook concluded from his experimental results that reserved words, parentheses, and brackets should also be considered as white space due to the extremely low error rate (and thus low information content) observed for these tokens. Entin decided that the definition of a software word was an open issue for research and considered all tokens, without exception, as words in her research.

For the current research, deletions were made based on Cook's definition of a word. The exact cloze procedure defined by Cook could not be used since it was designed for Pascal and the programs for the current experiments were coded in PL/I. Instead, Cook's cloze procedure was conceptually generalized to be language independent and then adapted to PL/I. A word was defined to be an identifier (name of a variable, constant, procedure, etc.), a constant (numeric or string with corresponding quote marks), or an operator (+, <=, WHILE, READ, etc).
"Punctuation" (e.g., colons, semicolons, commas, and parentheses), comments, and "reserved keywords" as defined by [HUGH79] were considered to be white space. Deletions were not made in comments since this was thought to confuse the issue of software cloze tests with prose cloze tests. Since all the literal strings used in the programs consisted of only one conceptually independent symbol, the issue of multi-word literal strings was not confronted in this research. Note, however, that the quote symbols used to delimit string literals were considered punctuation, but unlike other punctuation which was ignored, the quote symbols and corresponding character string were treated as a unit and if the literal were to be deleted, then the corresponding quotes were also deleted (e.g., 'TRUE' was counted as a single token and not a combination of a string token and a couple of quote mark tokens).

The cloze tests for the current experiments were constructed by replacing every fifth "word" of the programs with a blank line 20 spaces in length. Like Cook, the deletion process was arbitrarily started with the first token of each program. Because of the difference in lengths of the two program versions, the easy version had 137 deletions and the hard version had only 99. The considerable difference in the number of deletion items between the treatment conditions meant that time to complete the tests could not be used as a dependent variable in the experiment.

While Cook only used a verbatim scoring technique in his research, Entin used both a verbatim scoring technique and the more difficult, yet intuitively appealing synonymic scoring technique. Entin found that the two scoring techniques produced similar results, but felt that further research was needed to "substantiate the close relationship between the two scoring methods" [ENT184, p. 47]. Both scoring techniques were also used in the current cloze experiments to further explore the relationship between the two methods.

The verbatim technique for scoring the cloze tests was not concerned with the "intent" or meaning of a response and counted as incorrect any response that did not exactly match the original text. Because the
responses were not in machine-readable form, some interpretation of the responses was necessary before they could be compared to the original text. The verbatim technique was guided by the following rules for interpretation:

1. Apparent spaces in tokens other than character string constants were ignored.

2. Standard notational conventions were given their usual interpretation (e.g., a lower case letter 'b' with a line through it was interpreted as indicating a blank space, an upper case 'E' with a slash through it was interpreted as the PL/I logical conjunction operator (&), a caret (^) was interpreted as the PL/I logical negation operator, etc.). However, an equals sign with a slash through it was not counted as the "not equals" symbol.

3. Sequences of blanks within a character string constant were counted as only one blank.

4. Dashes in variable names were considered to be underscores due to the difficulty in differentiating between underscores and dashes.

5. All lower case letters were interpreted as upper case.

The synonymic scoring method used the same rules of interpretation as the verbatim technique, but instead of considering only exact matches as correct, the synonymic method allowed any syntactically correct response that preserved the input/output functionality of the program. The following syntactic "mistakes" were overlooked when considering the syntactic correctness of a token:

1. Character string constants with missing quotation mark delimiters.

2. Apparent spelling errors as determined by the experimenter.

3. Otherwise correct responses that included adjoining text that had not been deleted. The instructions for the cloze tests specified that each blank was to be filled-in with only one token, but some subjects seemed to get "carried away" and did not notice that they
were responding with more than one token and "filling-in" text that was not missing. The responses would have been correct if the extra text supplied by the subject had also been deleted from the original material.

4. Extraneous punctuation in format specifications.

5. An equals symbol with a slash through it as the PL/I "not equals" operator.

Other than by overlooking the above syntactic "mistakes", the synonymic scoring technique also differs from the verbatim scoring technique in the way it scores certain 100% entropy items. As always, the verbatim method considers a cloze item correct only if it is completed with the token supplied in the unabridged program. However, if all instances of an identifier (variable name, procedure name, etc.) are deleted from a program, then a subject has no way of determining the missing identifier from the remaining context. In such cases the synonymic technique scores as correct any syntactically correct identifier, not just the identifier supplied in the original program, if the supplied identifier does not coincide with any other identifier and it consistently replaces all instances of the missing identifier. If the replacement is not consistent, then the cloze items filled-in with the most frequently occurring identifier are scored as correct and the remaining are scored as incorrect (if no one identifier occurs more frequently than another, then one identifier may be arbitrarily chosen as correct and the others are incorrect).

Another case of 100% entropy is when all values assigned to or compared against a Boolean variable are missing. It is not the missing values themselves, but the relationship among the values, that is important for the proper functioning of the program. For example, assume a program contains three references to a Boolean variable, FLAG. Assume further that at one point FLAG is assigned the value TRUE, at another point it is assigned the value FALSE, and at the third reference FLAG is checked to see if it is equal to TRUE. The functioning of the program would not
be altered if the TRUE values were changed to FALSE and vice-versa. In some instances the actual values TRUE and FALSE could be changed to other values such as YES and NO, ONE and TWO, etc., without changing the functionality of the program. In these cases the synonymous scoring technique scores as correct the largest possible set of cloze items that is syntactically correct and that preserves the functionality of the program.

A third example of where the verbatim and synonymous scoring technique may disagree is language-dependent. The cloze unit in the PL/I statement

```
DECLARE MASTER_FILE ________ INPUT;
```

may be completed with either the token FILE or the token STREAM without changing the functionality of the statement. Thus, the synonymous technique would score either response as correct, but the verbatim technique would only score the token that matched the token selected by the author of the program. It is cases such as these previous examples that make the synonymous scoring technique more difficult to use and less likely to be computerized compared to the verbatim scoring technique.

The current cloze experiments exhibited two improvements over the Cook and Entin experiments. For one, the test programs used in the current experiments were less toy-like and more akin to real-world programs. While the Cook and Entin programs were approximately 30 lines long, the programs used in the current experiments were about 200 to 300 lines long. The shortness of the programs used by Cook and Entin only allowed approximately 30 deletions for their cloze tests. As previously reported, research within the prose domain suggests that cloze tests should have at least 50 deletions to obtain maximum measure sensitivity [BORM64a]. Without information to the contrary, it is probably wise to assume such a result also applies to the software domain. As a second improvement over the Cook and Entin experiments, the cloze tests used in the current research all contained at least 50 deletions.
Experiment 1

Design and Overview. A single-factor experiment was performed with the independent variable being program version (conventional or "easy", and encapsulated or "hard") and the dependent variable being the score on cloze tests. Score was defined as the percentage of cloze items completed correctly. As previously noted, time to completion was not used as a dependent variable in this experiment. The hypothesis of the experiment was that the cloze test results would indicate the same relative comprehensibility of the two programs as shown in the two pilot studies, that is, the average score on the cloze test for the easy version of the master file update program would be higher than the average score for the hard version of the program. An additional hypothesis was that the statistical results would hold regardless whether the cloze tests were scored using a verbatim or a synonymic technique.

Subjects. Twenty-two students in advanced level computer science courses at The Ohio State University participated in the experiment. All but two of the subjects were undergraduates. The subjects were paid volunteers solicited in the same manner and from the same population as in Pilot Study 2. Each subject was paid $5.00 an hour for a maximum of two hours of participation in the experiment. A signed experimentation consent form was obtained from each subject before the start of the experiment [see Appendix L].

Materials. The experimental materials used in this experiment were similar to the materials used in the second pilot study with the exception that the multiple-choice comprehension quizzes and the corresponding program listings were replaced with cloze tests that had been constructed from the two program versions in a manner previously described. Refer to Appendix V and Appendix W for listings of the easy program version cloze test and hard program version cloze test, respectively. The experimental instruction sheet [Appendix T] was the same as for the second pilot study except the multiple-choice quiz instructions were replaced with cloze test instructions. The program
description [Appendix U] for the two program versions was also the same as for the second pilot study except that input/output format information was added to enable the subjects to fill-in the deletions made in the I/O format sections of the cloze tests.

Procedure. The same general procedure was followed for Experiment 1 as for Pilot Study 2. The subjects were run in small groups and individually within a week of each other and at the convenience of their schedules. The subjects were balanced across treatment conditions based on their programming experience with PL/I as reported in the background surveys completed before the experiment [Appendix A]. Each subject was allowed an hour and forty-five minutes to complete the cloze test. As with the pilot studies, questions from the subjects were answered individually. There were few questions and the subjects seemed to understand their task of completing the cloze tests. On completion of the test a subject filled-in the post-experimental questionnaire, was paid, and was allowed to leave.

Results. All subjects completed the experiment in the allotted time. The reliability of the two cloze tests was analyzed using the Kuder-Richardson reliability estimate (KR-20). For the purposes of the reliability analysis, the cloze tests were scored using the synonymic scoring technique, the most widely accepted scoring method for cloze tests. The results of the analysis are reported in Table 9.

<table>
<thead>
<tr>
<th>Test (Program)</th>
<th>Kuder-Richardson Reliability Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>.51</td>
</tr>
<tr>
<td>Hard</td>
<td>.38</td>
</tr>
</tbody>
</table>

The reliability estimate of .51 for the cloze test constructed from the easy version of the program was satisfactory, but the low reliability estimate for the hard program test indicates that test had little inter-
item consistency. This result is similar to the one found in the reliability analysis performed for Pilot Study 2. An item analysis of the hard program cloze test revealed that 18 of the 99 test items were negatively correlated with total test scores (items number 8, 13, 14, 15, 26, 27, 42, 51, 52, 55, 58, 69, 70, 77, 78, 80, 82, and 97). Although the negative correlations were low and not significant (the "worst" was -.30), as in Pilot Study 2 an analysis was made to see if deleting some of the negatively correlated would improve the reliability of the measure. As few items were deleted as possible to minimize the effect the deletions would have on the score t-test. As expected, each deleted item slightly improved the reliability statistic. Table 10 summarizes the results from a Kuder-Richardson analysis after discarding from the hard program cloze test the four test items that had the largest negative correlation with total test score (items 13, 51, 77, and 97). Since the cloze tests for the two treatment conditions are not the same measure (as were the multiple-choice quizzes used in the pilot studies), no "corresponding" items were deleted from the easy program cloze test.

Table 10
Experiment 1 Reliability Analysis After Deletion from Hard Program Cloze Test of Four Items Negatively Correlated with Overall Score

<table>
<thead>
<tr>
<th>Test (Program)</th>
<th>Kuder-Richardson Reliability Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>.51</td>
</tr>
<tr>
<td>Hard</td>
<td>.50</td>
</tr>
</tbody>
</table>

Inspection of Table 10 reveals that after the removal of the four items from the hard program cloze test, the reliability of the hard program test increased to an acceptable value. Probable causes for the low Kuder-Richardson reliability estimates will be discussed in the upcoming "Discussion" section.

The score experimental results are summarized in Tables 11 and 12. Since the number of items differed for the two program versions, the
scores are expressed as percentages of the number of items. Table 11 shows the analysis of the measures with the entire set of items and Table 12 shows the analysis with the four items removed from the hard program test. With or without the negatively correlated items, one-tailed t-tests revealed the same significant score effect as found in the two pilot studies. Subjects working with the cloze test constructed from the easy version of the program had significantly higher scores than the subjects working with the cloze test constructed from the hard version of the program. The tables also show that the results were the same regardless of the technique used to score the tests. The scores produced by the two different scoring strategies were highly correlated ($r = .95, p < .001$).

Table 11

Experiment 1 t-test Analysis of Percent Score

<table>
<thead>
<tr>
<th>Scoring Technique</th>
<th>Easy Program</th>
<th>Hard Program</th>
<th>t</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonymic</td>
<td>M 96.6</td>
<td>88.5</td>
<td>7.99</td>
<td>.001</td>
</tr>
<tr>
<td></td>
<td>SD 1.6</td>
<td>2.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verbatim</td>
<td>M 95.3</td>
<td>84.7</td>
<td>7.44</td>
<td>.001</td>
</tr>
<tr>
<td></td>
<td>SD 2.7</td>
<td>3.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*aNumber correct items divided by total number items.

*b$n = 11$ for each group.

Table 12

Experiment 1 t-test Analysis of Percent Score After Deletion from Hard Program Cloze Test of Four Items Negatively Correlated with Overall Score

<table>
<thead>
<tr>
<th>Scoring Technique</th>
<th>Easy Program</th>
<th>Hard Program</th>
<th>t</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonymic</td>
<td>M 96.6</td>
<td>90.0</td>
<td>5.66</td>
<td>.001</td>
</tr>
<tr>
<td></td>
<td>SD 1.6</td>
<td>3.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verbatim</td>
<td>M 95.3</td>
<td>86.0</td>
<td>7.19</td>
<td>.001</td>
</tr>
<tr>
<td></td>
<td>SD 2.7</td>
<td>4.6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*aNumber correct items divided by total number items.

*b$n = 11$ for each group.
Responses to the background survey and post-experimental questionnaire were correlated with the independent and dependent variables. Because of the similarity between the scores as determined by the two scoring methods, the correlations are only performed with the synonymic scores. The means and standard deviations of the responses are reported in Appendix X and the correlations are reported in Appendix Y. As with the pilot studies, to help avoid spurious correlations only significant correlations that hold for each treatment group as well as for the combined groups will be reported for an experiment.

There was no significant correlation between the number of PL/I programs written by a subject and the treatment group in which the subject was placed. The mean number of PL/I programs written by each group were also similar (30.1 for the easy program version group and 25.4 for the hard program version group). These results suggest that subjects were properly balanced across treatment conditions according to the balancing criterion. There were no background survey or post-experimental questionnaire variables that were significantly correlated with the dependent variable for both the individual treatment groups and the combined groups.

Discussion. Experiment 1 was a conceptual replication of Pilot Study 2 using a different subject pool and a cloze test measure instead of a multiple-choice quiz measure. The hypothesis that the conventional version of the program was easier to comprehend than the encapsulated version of the program was supported by the results of the cloze tests. The hypothesis that the experimental results would hold using both the synonymic and verbatim scoring techniques was also supported.

The reliability of the cloze tests was shown to be acceptable, but only after four items that were negatively correlated with total test score were removed from the cloze test for the hard program version. Due to the larger number of items in the cloze tests than in the pilot study multiple-choice quizzes (about a 7-to-1 increase), one would expect the reliability of the cloze tests to be higher than the reliability of the multiple-choice quizzes. Instead, the average of the reliability
estimates calculated for the multiple-choice quizzes (Tables 3 and 6) was .58 and the average of the reliability estimates for the cloze tests (Table 10) was .50. A possible reason for this discrepancy is that a large percentage of the items on the cloze tests were correctly completed by all subjects. As previously mentioned, reliability is a function of inter-subject variability. The larger the percentage of items on a test correctly completed by all subjects, the lower the variability in scores possible among the subjects. For the easy and hard program cloze tests, 82% and 53%, respectively, of the items were completed correctly by all the subjects (there was probably a ceiling effect for the easy program cloze test). In contrast, there were no multiple-choice items that were completed correctly by all subjects. It is believed the reason the initial reliability estimate for the hard program cloze test was lower than the reliability estimate for the easy program cloze test was that more of the hard program cloze test items than the easy program cloze test items violated the assumption of the Kuder-Richardson formula that the items measure the same function as the test overall. Eighteen of the 99 hard program cloze test items (18%) were negatively correlated with overall test score while only 8 of the 137 easy program cloze test items (6%) were negatively correlated with overall score. However, since all the negative correlations were low (none were even slightly significant), it was hoped that the many negatively correlated items for the hard program cloze test was only a chance occurrence and that it did not indicate a trait of software cloze tests. If it were only a chance event, then the reliability estimates for the cloze tests used in the next experiments should be much higher (if ceiling and floor effects can be avoided).

The cloze test scores found in this experiment were not typical of cloze test scores found by prose researchers. Prose researchers often encounter cloze scores of about 50%. Bormuth [BORM71] concluded from his research on prose passage reading performance criteria that cloze scores of 57% and above are at the high end of reading performance. The cloze scores found in this experiment were about 90%. Although these scores are in the software domain and may not have the same
characteristics as scores in the prose domain, they still seem high. The many items correctly completed by all subjects and the high cloze scores suggest that the current cloze tests were "too easy". To corroborate the cloze test results found in Experiment 1, a second cloze experiment was performed using the same program versions as in Experiment 1.

Experiment 2

Experiment 2 was a replication of Experiment 1 with a modified cloze procedure. In an attempt to lessen the number of cloze items successfully completed by all subjects (and thus potentially improve the reliability of the cloze tests) a deletion rate greater than the 1-in-5 rate used in Experiment 1 was selected for Experiment 2. A higher deletion rate leaves less contextual information for completing the deletions and this should make the cloze items more difficult to complete, increasing the error rate. A deletion rate of 1-in-4 did not seem significantly different from a rate of 1-in-5 and a rate of 1-in-2 seemed too difficult, so an in-between deletion rate of 1-in-3 was selected as a compromise. Since increasing the deletion rate to 1-in-3 was thought to generate too many cloze items (almost 230 for one program version), the number of cloze items was reduced by not making any deletions in variable declaration statements. Considering that the score and time effects found in the pilot studies resulted from multiple-choice quizzes that did not contain questions directly concerned with variable declaration statements, it was felt that cloze tests which also ignored variable declaration statements should still indicate comprehensibility differences between the two program versions. A one-tailed t-test performed on Experiment 1 cloze data that had been modified by excluding deletions in the variable declaration statements showed the same significant score effect found in Experiment 1 [see Table 13]. Although this post-experimental result is no guarantee that the Experiment 1 score effect would have been found if the experiment actually had been conducted without deletions in the variable declaration statements, it suggests such a result.
Table 13
Experiment 1 t-test Analysis of Percent Score* Ignoring Cloze Items in Variable Declaration Statements

<table>
<thead>
<tr>
<th>Scoreb</th>
<th>Easy Program</th>
<th>Hard Program</th>
<th>t</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>92.6</td>
<td>87.6</td>
<td>2.68</td>
<td>.01</td>
</tr>
<tr>
<td>SD</td>
<td>7.6</td>
<td>13.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*aNumber correct items divided by total number items. Synonymic scoring.
b \( n = 11 \) for each group.

The purpose of Experiment 2 was to obtain additional evidence that a cloze procedure could be used to measure program comprehensibility. It was hoped that the reliability of the cloze tests used in the current experiment would also be higher than the reliability of the cloze tests used in Experiment 1.

Design and Overview. As in Experiment 1, a single-factor experiment was performed with the independent variable being program version ("easy" or conventional, and "hard" or encapsulated) and the dependent variable being the percentage of cloze items completed correctly. Time to completion was again not used as a dependent variable because of the difference in the number of cloze items for each treatment group (82 for the easy program version and 95 for the hard). The hypothesis of the experiment was that the cloze test results would indicate the same relative comprehensibility of the two programs as shown in the two pilot studies and the previous experiment, that is, the average percentage of items completed correctly on the cloze test for the easy version of the master file update program would be higher than the average percentage of cloze items completed correctly on the cloze test for the hard version of the program. An additional hypothesis was that the statistical results would hold regardless whether the cloze tests were scored using a verbatim or a synonymic technique.

Subjects. Twenty-two students in advanced level computer science courses at The Ohio State University participated in the experiment.
All the subjects were undergraduates. The subjects were paid volunteers solicited in the same manner and from the same population as in Experiment 1 and Pilot Study 2. Each subject was paid $5.00 an hour for a maximum of two hours of participation in the experiment. A signed experimentation consent form was obtained from each subject before the start of the experiment [see Appendix L].

Materials. The experimental materials used in this experiment were the same as the materials used in Experiment 1 with the exception that the cloze tests constructed from the two versions of the master file update program had 1-in-3 tokens deleted instead of 1-in-5 and no tokens were deleted from variable declaration statements. Refer to Appendix Z and Appendix AA for listings of the easy program version cloze test and hard program version cloze test, respectively. The experimental instruction sheet [Appendix T] and the program description [Appendix U] for the two program versions were the same as in Experiment 1.

Procedure. The same general procedure was followed for Experiment 2 as for Experiment 1. The subjects were run in small groups and individually within a week of each other and at the convenience of their schedules. The subjects were balanced across treatment conditions based on their programming experience with PL/I as reported in the background surveys completed before the experiment [Appendix A]. Each subject was allowed an hour and forty-five minutes to complete the cloze test. There were few questions and the subjects expressed no difficulty in understanding the task of completing a cloze test. On completion of the test a subject filled-in the post-experimental questionnaire, was paid, and was allowed to leave.

Results. All subjects completed the experiment in the allotted time. The reliability of the two cloze tests was analyzed using the Kuder-Richardson reliability estimate (KR-20). As with Experiment 1, for the reliability analysis, the cloze tests were scored using the synonymous scoring technique. The results of the analysis, reported in Table 14, indicate that both of the cloze tests had high inter-item consistency.
Table 14
Experiment 2 Reliability Analysis

<table>
<thead>
<tr>
<th>Test (Program)</th>
<th>Kuder-Richardson Reliability Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>.88</td>
</tr>
<tr>
<td>Hard</td>
<td>.97</td>
</tr>
</tbody>
</table>

The score experimental results are summarized in Table 15. Recall that since the number of items differed for the two program versions, the scores are expressed as percentages of the number of items on a test. Unlike Experiment 1, one-tailed t-tests did not reveal any significant score effect. Subjects working with the cloze test constructed from the easy version of the program scored the same as the subjects working with the cloze test constructed from the hard version of the program. Table 15 also shows that the results remained the same regardless of the technique used to score the tests. As in Experiment 1, the scores produced by the two scoring strategies were highly correlated ($r = .99, p < .001$).

Table 15
Experiment 2 t-test Analysis of Percent Score*

<table>
<thead>
<tr>
<th>Scoring Technique</th>
<th>Easy Program Score*</th>
<th>Hard Program Score*</th>
<th>t</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonymic</td>
<td>M 87.7, SD 7.6</td>
<td>M 85.7, SD 13.3</td>
<td>.42</td>
<td>.34</td>
</tr>
<tr>
<td>Verbatim</td>
<td>M 86.6, SD 7.9</td>
<td>M 84.2, SD 13.9</td>
<td>.49</td>
<td>.31</td>
</tr>
</tbody>
</table>

*Number correct items divided by total number items.

Responses to the background survey and post-experimental questionnaire were correlated with the independent and dependent variables. Because of the similarity between the scores as determined by the two scoring methods, the correlations are only performed with the synonymic scores. The means and standard deviations of the responses are reported in
Appendix B and the correlations are reported in Appendix C. As with the previous analyses of the background and post-experimental responses, to help avoid spurious correlations only significant correlations that hold for each treatment group and the combined groups are reported for an individual experiment.

There was no significant correlation between the number of PL/I programs written by a subject and the treatment group in which the subject was placed. The mean number of PL/I programs written by each group were also similar (23.1 for the easy program version group and 21.1 for the hard program version group). These results suggest that subjects were properly balanced across treatment conditions according to the balancing criterion. The only background survey or post-experimental questionnaire variable that was significantly correlated with the dependent variable for both the individual treatment groups and the combined groups was the age of the subject. The older the subject, the lower the subject scored (for the combined groups, \( r = -0.82, p < 0.001 \)).

This result is probably due to two facts: (1) the oldest subject in each group was also the poorest scorer for the group and (2) the distance in age and score between the oldest subject in each group and the rest of the group was large compared to the ranges of the ages and scores for the rest of the group. When the two oldest subjects are removed from the analysis, the combined group correlation is no longer negative or significant.

Discussion. Experiment 2 was a conceptual replication of Experiment 1 using a modification of the Experiment 1 cloze procedure. The hypothesis that the conventional version of the program was easier to comprehend than the encapsulated version of the program was not supported by the results of the cloze tests. Neither the synonymic or verbatim scoring techniques produced a significant score effect. The reliability of each of the cloze tests was high, an improvement over the reliability of the Experiment 1 cloze tests (from an average of 0.50 in Experiment 1 to an average of 0.92 in Experiment 2). The increase in the reliability may be due to the reduction from Experiment 1 in the
percentage of cloze items on each test that were correctly completed by all subjects (from 82% to 51% for the easy program cloze test and from 53% to 35% for the hard program cloze test). The large percentage of test items negatively correlated with total score found for the hard program cloze test in Experiment 1 did not occur again in Experiment 2 (from 18% to 7%). Also, the probable score ceiling effect for the easy program cloze test of Experiment 1 was eliminated in Experiment 2 (from 97% to 86%). As in Experiment 1, the cloze test scores for Experiment 2 were high in comparison to cloze tests scores typically encountered in the prose domain. While cloze scores in the prose domain are around 50%, the scores in Experiment 2 were about 86%.

Of key interest is that the results from Experiment 1 agreed with the results from the two pilot studies (i.e., that the encapsulated version of the master file update program is more difficult to comprehend than the conventional version) while the results from Experiment 2 did not agree. There are several possibilities that would explain the discrepancy between the results of the two experiments. Given that both of the experiments used subjects from the same population, that both experiments used similar materials and had similar experimental conditions, that subjects were balanced across treatment conditions in the same manner, that all subjects completed the experiments, and that none of the subjects expressed any difficulty in understanding the cloze task, it was assumed that the difference in the outcomes of the experiments did not result from faults in the design or procedure of one or both of the experiments. Instead, the focus of the analysis will be concentrated on the differences in the treatment conditions of the two experiments.

There were three major differences in the tests materials for the two experiments: (1) the deletion rate (from 1-in-5 to 1-in-3), (2) the program sections in which deletions were made (deletions were made in variable declaration statements in Experiment 1 and not in Experiment 2), and (3) the actual deletions made to construct each cloze test. Concerning the first difference, it was expected that increasing the cloze test deletion rate from 1-in-5 for Experiment 1 to 1-in-3 for
Experiment 2 would also increase the difficulty of the cloze tests and that the test scores would be lowered. If the cloze tests had become too difficult, then differences in the comprehensibility of the two program versions could have been masked by the difficulty of the tests themselves. That is, the subjects could have been responding more to the incomprehensibility of the cloze tests than to the comprehensibility of the program versions. However, given that the average score for Experiment 2 was above 80%, it does not seem that the cloze tests were too difficult and it is not felt that this difference in the experimental materials explains the discrepancy between the outcomes of the experiments.

As previously argued, it was expected that excluding deletions from variable declaration statements in the Experiment 2 cloze tests would not affect the relative difficulty of the tests. One-tailed t-tests of the Experiment 1 data which had been modified by excluding variable declaration statement cloze items still indicated the same relative comprehensibility of the two program versions as had the unmodified data. Such a post-experimental analysis does not, of course, reveal how subjects would have responded in Experiment 1 if deletions had not been made in the variable declaration statements, but neither does it suggest that the subjects would have responded in a substantially different manner. Although it may not be true for all programs, it is intuitively appealing that, for the programs used in these experiments, deletions in the variable declaration statements should not affect the relative difficulty of the corresponding cloze tests. It is intuitively appealing because the data structures used in the respective programs were virtually identical to each other and the manner in which the data structures were manipulated by the programs seemed to be of more importance than the manner in which the data structures were declared. Also, the multiple-choice quizzes used in the pilot studies indicated the expected relative comprehensibility of the two programs without any of the multiple-choice questions directly referring to the variable declaration statements. It was therefore considered that if the cloze tests were properly measuring program comprehension, then whether
deletions were made in variable declaration statements should not have affected the experimental outcome.

An assumption underlying most uses of cloze testing is that a cloze test is a "stable" measure. If the results of the current experiment had occurred using the same subjects in the same treatment conditions as in Experiment 1, then it would have been concluded that the cloze tests did not have alternate-form reliability. By a "stable" measure it is meant that all cloze tests constructed from the same test materials and based on the same cloze procedure should produce the same results with similar groups of subjects. For example, assuming all else is equal, subjects working with a 1-in-5 cloze test with deletions beginning at the first word of the test materials should respond the same as similar subjects working with a 1-in-5 cloze test with deletions beginning at the third word. Deleting from the test material "enough" times is supposed to ensure that all cloze tests represent the original material. Thus, no thought was given to which pattern of deletions was used to construct the cloze tests for Experiments 1 and 2 and an arbitrary choice was made to begin all the deletions with the first token of each program. However, as noted in Chapter II, some researchers of cloze testing in the prose domain have suggested that different every-nth patterns can differentially affect performance because some patterns may have different characteristics than others and may not represent the material being measured [BOYCE82, ENTI78, MERE78, PORT78, BORM64b].

Although every-nth deletions are thought to approximate random deletions in the long run, this may not be true for materials which are potentially cyclic in structure. Studies of the structural properties of programs such as Knuth's Fortran study reveal that programs written in Fortran (and Fortran-like languages such as PL/I) are highly regular in nature [KNUT71]. Knuth found that (Fortran) assignment statements are commonly of the form "A = B". An equivalent form in PL/I would be "A = B;". The cloze procedures used in this research consider both the assignment statements as consisting of three tokens. Since assignment statements are usually the most frequently occurring statement in a
program and often occur in clusters, it is easy to see that a 1-in-3 deletion strategy could produce a biased sample by deleting the same element from each of a group of assignment statements. This author has also noticed that the 1-in-5 deletion strategy used in the current research would get "in sync" with the declarations of variables in PL/I programs and delete the same information from a group of declarations. This phenomenon is demonstrated by the following program segment taken from Appendix V.

```
DECLARE 1 TRANS_REC
  2 TRANS_CODE, FIXED(11,-------------------1),
  2 TRANS_INFO,-------------------
  3 TRANS_KEY,-------------------(4,0),
  3 TRANS_AMOUNT,-------------------(7,2),
  3 TRANS_NAME,-------------------(35),
  3 TRANS_ADDRESS CHAR(-------------------);
DECLARE 1 OLD_MAST_REC
  2 STATIC, FIXED(4,0),
  2 ------------------- FIXED(7,2),
  2 OCD_MAST_ADDRESS CHAR(35),
DECLARE 1 NEW_MAST_REC
  2 STATIC,-------------------
  2 NEW_MAST_KEY FIXED(4,0),
  2 ------------------- NEW_MAST_BALANCE FIXED(7,2),
  2 ------------------- NEW_MAST_NAME CHAR(35),
DECLARE CURRENT_KEY STATIC (4,0);
DECLARE IN_USE STATIC (3);```

Just as with the prose domain, it may be true in the software domain that different every-nth patterns can differentially affect performance because of differences in the types of items deleted. A possible explanation for the discrepancy in the outcomes of Experiments 1 and 2 is that the cloze tests used in Experiment 1 were representative of the two programs (and thus demonstrated the proper difference in comprehensibility) and the cloze tests used in Experiment 2 were not representative (and did not demonstrate the difference).

It is not known how to determine if a given cloze test represents the material from which it is constructed without first administering the test. However, it may be possible to compare the characteristics of the cloze tests used in Experiments 1 and 2 and explain the differences in the outcomes of the experiments by differences in the characteristics of the cloze tests. To explore this possibility, a post-experimental analysis was performed on the cloze tests used in Experiments 1 and 2.
CHAPTER V

Analysis of the Experiment 1 and Experiment 2 Cloze Test Characteristics

The difficulty in interpreting the results of Experiments 1 and 2 was that the outcome of one experiment suggested that there was a difference in the comprehensibility of two programs while the outcome of the other experiment suggested that the comprehensibility of the two programs was the same. Since the results from the two experiments seemed trustworthy, it was concluded that the cloze tests were not properly measuring program characteristics (which did not change between the two experiments) and that the discrepancy in the outcomes of the two experiments was due to differences in the cloze tests. It was argued in Chapter IV, however, that the differences in the cloze tests due to variations in the cloze procedures used in the two experiments (deletion rate and deletions made in variable declaration statements) did not seem to explain the discrepancy. The remaining explanation for the discrepancy was the possibility that cloze tests produced using an every-nth deletion pattern may not be representative of the test materials from which they are constructed.

It was suggested in Chapter IV that there may be differences in the characteristics of two every-nth cloze tests constructed from the same material. It was reasoned that the regularities in the test material from which a cloze test is constructed may get in sync with the regularities of an every-nth deletion pattern and that the resulting set of cloze items may not form a representative sample of the test material. If different cloze tests sample different aspects of some test material, then it might be expected that the outcomes of the cloze tests will be different. One way to check to see if different cloze tests are representative of some test material is to see if they have different characteristics. If they do have different characteristics, then they cannot all be representative.
Experience with developing software cloze tests demonstrated to this author that one characteristic of cloze tests is that they may contain two types of cloze items. To be successfully completed, certain cloze items required at least some understanding of the purpose, functionality, algorithm, etc., of the program in which the cloze items were embedded. Other cloze items could be completed without any such understanding of the program. Instead, these cloze items could be completed using only syntactic knowledge and general reasoning skills. The following is an example of a cloze item that may be completed using only syntactic knowledge and without any significant comprehension of the program in which it was embedded:

```
TOTAL _____ 0;
```

Given that the completed cloze item is to be syntactically correct, it can be correctly completed without knowledge of the semantics of the corresponding PL/1 program because the syntax rules of PL/1 allow only an equals sign in such a context. A less simple example of a cloze item that may be correctly completed without having to comprehend the semantics of the corresponding program is contained in the following statement:

```
CALL NEXT_KEY(_____,MAST_KEY,CURRENT_KEY);
```

If elsewhere in the program containing the above statement there is a PROCEDURE statement such as

```
PROCEDURE NEXT_KEY(TRANS_KEY,MAST_KEY,CURRENT_KEY);
```

then it is likely that the missing parameter is TRANS_KEY, since it is common for programmers to represent actual and corresponding dummy parameters with the same identifier when possible.

The above example cloze items could be completed by rules, knowledge, or heuristics that are independent of the semantics of the corresponding programs. Such cloze items will be termed "program-independent". Cloze items that are not program-independent will be termed "program-dependent". An example of a program-dependent cloze item is contained
in the following program (arrow points to statement containing cloze item):

```plaintext
EXAMPLE:PROCEDURE OPTIONS(MAIN);
DECLARE NUM  FIXED(3,0);
DECLARE COUNT FIXED(3,0);
DECLARE SUM   FIXED(9,2);
DECLARE AVG   FIXED(7,2);
DECLARE MORE_DATA CHARACTER(5);
ON ENDFILE(SYSIN) MORE_DATA = 'FALSE';
MORE_DATA = 'TRUE';
SUM  = 0;
COUNT = 1;
GET LIST(NUM);
DO WHILE(MORE_DATA = 'TRUE');
   SUM = SUM + NUM;
   COUNT = COUNT + 1;
   GET LIST(NUM);
END;
   --> COUNT = COUNT + 1;
   AVG = SUM/COUNT;
   PUT LIST('AVERAGE = ',AVG);
END EXAMPLE;
```

Although PL/I syntax may dictate that the missing token must be an operator, the cloze item is program-dependent because the determination of which operator correctly completes the item requires knowledge concerning the function of the program and the workings of the algorithm used to achieve that function.

The significance of program-independent and program-dependent categories of cloze items is that they can be used to explain the discrepancy between the results of Experiments 1 and 2. The explanation is built on three assumptions:

1. For a given program, different every-nth deletion patterns may have different proportions of program-independent and program-dependent cloze items,

2. For comparable cloze tests, program-dependent cloze items have a higher error rate and take more time to complete than program-independent items, and
the easy and hard program versions used in Experiments 1 and 2 are syntactically and semantically similar enough to each other to make it sensible to compare the characteristics of cloze items from each version (i.e., the two versions are comparable).

Applying assumption 1, it is possible that in Experiment 1 the easy program cloze test had a smaller proportion of program-dependent cloze items than the hard program cloze test while in Experiment 2 the two cloze tests had a comparable proportion of program-dependent cloze items. Applying assumptions 2 and 3, the discrepancy in the results of the two experiments may be because having a smaller proportion of difficult cloze items should make the easy program cloze test easier than the hard program cloze test (Experiment 1) and having a similar proportion of difficult cloze items should make the cloze tests have the same difficulty.

All three of the above assumptions have intuitive appeal. It has previously been argued that different every-nth deletion patterns can get in sync with a program's syntax and thus generate sets of cloze items with different characteristics (the first assumption). Also, given that a subject completing a cloze test is experienced in the cloze test implementation language, it seems reasonable to assume that the program-independent cloze items should have a lower error rate and take less time to complete than the program-dependent items (the second assumption). The rationale behind this is that the solutions to the program-independent cloze items, by definition, may be arrived at by using knowledge that should already be understood by the test subject before attempting to complete the test. In general, the completion of the program-dependent cloze items should require the same knowledge as for program-independent items plus additional program-dependent knowledge that must be synthesized from the cloze test. The gathering and application of this extra program-dependent knowledge should make
the program-dependent cloze items more difficult to solve as indicated by
error rates and time needed for completion. Concerning the third
assumption, it is not known how similar two programs must be to make it
reasonable to compare the characteristics of cloze items generated from
each program. Certainly, though, the more dissimilar the two programs,
the more difficult it is to compare cloze tests constructed from the
programs because of the possible confounding effect of differences in
the programs. But, as has been previously discussed in Chapter III,
efforts were taken to make the two programs comparable across most
dimensions. Both of the programs were based on the same algorithm, were
functionally equivalent in the sense that the input for the two programs
was identical and that identical proper input for each program would
produce identical output, were of similar length, used similar operators
and operands, etc. Thus, it should be reasonable to compare the cloze
tests constructed from the programs.

To verify the assumptions made concerning program-independent and
program-dependent cloze items it is, of course, necessary to be able to
differentiate between the two categories of items. The following list
of guidelines was used to define program-independent cloze items. Cloze
items not covered by the list were considered to be program-dependent.
Note that there is no claim that the guidelines are in any sense
complete since the guidelines were developed based only on the cloze
items encountered in the cloze tests constructed for the current
research. Also, all the test programs on which the cloze tests were
based were written in a single programming language (PL/I) and solved
the same problem. There may be program-independent methods for
determining the solution to particular cloze items that have gone
undiscovered even for the cloze items used in the current research. The
purpose of the guidelines was to show how different every-nth deletion
patterns have different characteristics and was not meant to provide a
definitive categorization of program-independent and program-dependent
cloze items.
Guidelines for PL/I Program-Independent Cloze Items

A PL/I cloze item is considered to be program-independent if the application of one or more of the upcoming guidelines results in the correct answer for the cloze item. If following the guidelines does not lead to a particular answer or if it leads to an incorrect answer, then the cloze item is considered to be program-dependent.

The first three guidelines are general in nature and are applicable to all cloze items:

1. The missing token must be syntactically correct. In cases where there is only one syntactically correct answer or where all the syntactically correct answers are also semantically correct, this rule leads directly to a correct answer. Usually this rule defines a set of candidate answers instead of a specific answer.

An example of a cloze item with only one syntactically correct answer is a missing assignment operator in an assignment statement. An example of a cloze item with more than one syntactically correct answer, but all the syntactically correct answers are also semantically correct is a missing variable name where all other references to the variable are also missing. Here, the missing variable name usually can be any legal identifier not already in use.

2. If all but one of a set of candidate answers has been eliminated by various other guidelines, then the remaining candidate answer is the desired solution.

3. The solution to a cloze item may be specified by a comment statement corresponding to the statement containing the cloze item.

Specific Categorization Rules

The rigid structure of PL/I (and most other programming languages) usually allows one to determine the syntactic characteristics of a given cloze item. It can generally be determined if a missing token is a variable name, procedure name, arithmetic operator, etc. The following
guidelines are more specific in nature and are grouped according to the syntactic characteristics of the missing token.

**Name on a PROCEDURE statement.**

1. Candidate answers are the procedures CALLeD in the program in the scope of the current procedure. Since the name must be unique, all CALLeD procedures with known corresponding PROCEDURE statements may be eliminated from the set of candidate answers.

2. May be specified by the name on the corresponding END of procedure statement.

3. May be determined from a corresponding CALL statement by matching the parameter list of the PROCEDURE statement to the parameter lists of all the CALL statements. The parameters should match in number, order, type, and (often) name.

**Name on an END of procedure statement.**

1. May be specified by the name on the corresponding PROCEDURE statement.

**Element in a DECLARE statement (e.g., variable name, level number, storage type, data type, precision, etc.).**

1. May be specified in another DECLARE for the same variable (matched by variable name or common substructure).

2. May be specified by way variable is used (e.g., the INPUT attribute is indicated for a file if it is used only in input statements).

3. May be specified by the I/O format specifications for the variable (e.g., a format specification of F(1) suggests the variable is FIXED and has precision (1,0)).

4. The candidates for a missing variable name are all the variables referenced in the corresponding procedure, but not already DECLARED.

**Format element (e.g., data type and precision).**

1. May be specified by the DECLARE statement for the corresponding variable (e.g., the DECLARE specification FIXED indicates the F format data type specification).

2. May be indicated in another format specification for the same variable.

**Procedure name in a CALL statement.**

1. Candidate answers are all subroutine procedures within the scope of the CALL statement.

2. May be determined from a corresponding PROCEDURE statement by matching the parameter list of the CALL statement to the parameter lists of all the PROCEDURE statements. The parameters should match in number, order, type, and (often) name.

3. All subroutine procedures should be CALLeD at least once.
Actual and dummy parameters.

1. Candidate answers are limited to the variables DECLARED in the corresponding procedure. All referenced parameters must be in the corresponding PROCEDURE statement parameter list.

2. May be specified in the parameter list of the corresponding CALL or PROCEDURE statement.

3. If an actual and corresponding dummy parameter cannot have the same name (because the name is not declared for both cases), then the parameter may be determined by matching the types of the parameters.

4. All variables DECLARED with an unspecified length (indicated by a '£') must be a dummy parameter.

Name of an unreferenced substructure of a structured variable in a DECLARE statement.

1. May be any syntactically correct variable name that is not the same as any other identifier not itself a part of the substructure.

Identifier in an assignment statement or conditional expression.

1. Candidate answers consist of all variables DECLARED in the corresponding procedure. All DECLARED variables should be referenced at least once (except for substructure variables). All dummy parameters should be referenced at least once.

2. A variable should not be assigned to itself or subtracted from itself (i.e., there should be no statements such as A = A; or A = B - B;).

3. Both sides of an assignment statement and conditional expression should be of compatible type, structure, and precision. The common description of both sides of the assignment statement or conditional expression should also be compatible (e.g., if the candidate answers for the cloze item in the statement

   _____ = '102 N. HIGH ST.';

are ADDRESS and NAME, then ADDRESS should be the answer since '102 N. HIGH ST.' would commonly be described as an address and not as a name).

File name on an ON ENDFILE statement.

1. Candidate answers are limited to variables DECLARED as files in the corresponding procedure.

2. There should be only one ON ENDFILE statement for any given file used within a procedure.

3. Should be same as file name on associated input statement.

ON condition for a file.

1. Should be ENDFILE condition if file used for input in corresponding procedure.

Variable in an I/O list.

1. DECLARED attributes of variable should be compatible with corresponding format specifications.
File name in an I/O statement.

1. Candidate answers are limited to variables DEClARED as files in the corresponding procedure.

2. Attributes of file from DECLARE statement (e.g. INPUT, OUTPUT, STREAM, RECORD, etc.) must match way file is used.

Using the above guidelines, the experimenter divided the cloze items from each of the cloze test used in Experiments 1 and 2 into program-independent and program-dependent categories. Appendix DD lists which items of the cloze tests used in Experiments 1 and 2 were categorized as program-dependent (items not listed, of course, were program-independent). Once categorized, the characteristics of the experimental cloze tests were then explored based on the categories.

Table 16 shows the proportion of program-dependent cloze items for each of the experimental cloze tests. An examination of Table 16 reveals that, as expected, different every-nth deletion patterns can generate different proportions of program-independent and program-dependent cloze items for the same program. For example, there was a 233% increase in the proportion of program-dependent cloze items in the easy program cloze tests in Experiments 1 and 2. Table 16 also shows that the proportion of program-dependent cloze items in the hard program cloze test in Experiment 1 was 133% larger than the proportion of program-dependent cloze items in the easy program cloze test in the same experiment. In contrast, in Experiment 2 the proportion of program-dependent cloze items in the hard program cloze test was 3% smaller than the proportion of program-dependent cloze items in the easy program cloze test. This large difference in the proportion of program-dependent cloze items for the two tests in Experiment 1 and small difference in the proportion of program-dependent cloze items for the two tests in Experiment 2 support the hypothesis that different every-nth cloze patterns constructed from the same program may have different characteristics and is one component of the explanation for the discrepancy found in the results of the two experiments.
Table 16
Proportion of Program-Dependent Cloze Items
for Experiments 1 and 2 Cloze Tests

<table>
<thead>
<tr>
<th>Program Version</th>
<th>Experiment 1</th>
<th>Experiment 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>11.7</td>
<td>39.0</td>
</tr>
<tr>
<td>Hard</td>
<td>27.3</td>
<td>37.9</td>
</tr>
</tbody>
</table>

Another component of the explanation for the discrepancy has to do with the relative difficulty of program-dependent and program-independent cloze items. Table 17 contains the difficulty of the program-dependent, program-independent, and combined categories of cloze items for each of the cloze tests in Experiments 1 and 2. The difficulty of a category for a particular cloze test was calculated as the number of times the cloze items within the category were completed incorrectly (as determined by the synonymic scoring method) divided by the product of the number of cloze items within the category and the number of subjects working with the test. The data in Table 17 support the assumption that, for comparable programs, program-dependent cloze items are more difficult than program-independent cloze items. In all cases the difficulty of the program-dependent cloze items is at least two times the difficulty of the program-independent cloze items.

Table 17
Difficulty of Different Categories of Cloze Items in Experiments 1 and 2

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Cloze Item Category</th>
<th>Program Version</th>
<th>Program-Dependent</th>
<th>Program-Dependent</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Easy</td>
<td>1.4</td>
<td>19.3</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>8.5</td>
<td>19.5</td>
<td>11.5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Easy</td>
<td>3.6</td>
<td>25.9</td>
<td>12.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>8.2</td>
<td>24.2</td>
<td>14.3</td>
<td></td>
</tr>
</tbody>
</table>
If it is proper to compare the characteristics of the cloze items in the various cloze tests, the combined data of Tables 16 and 17 support the notion that the reason the expected score difference was found in Experiment 1 and not in Experiment 2 is that the easy program cloze test in Experiment 1 had a lower percentage of program-dependent (difficult) cloze items than did the hard program cloze test while the two cloze test in Experiment 2 had the same percentage of program-dependent cloze items.

Another way to examine the influence of program-dependent cloze items is to compare the same cloze test between experiments (an inter-experimental analysis in contrast to the previous intra-experimental analysis). It was expected that increasing the deletion rate from 1-in-5 in Experiment 1 to 1-in-3 in Experiment 2 would also increase the cloze item error rates because of the decrease in the context available to aid filling-in the cloze items. Comparing the Experiment 1 synonymic error rates to the Experiment 2 error rates (derived from Tables 11 and 15 of Chapter IV by subtracting the synonymic scores from 100) reveals that the error rates for the easy and hard program cloze tests did increase, but not in a uniform manner. The easy program error rate increased by a large percentage while the hard program error rate only increased by a small percentage. Similarly, the proportion of program-dependent cloze items in the easy program cloze tests increased by a large percentage while the proportion of program-dependent cloze items in the hard program cloze tests only increased by a small percentage (refer to Table 18). Thus, an inter-experimental analysis as well as an intra-experimental analysis support the notion that the different proportions of program-dependent cloze items in the Experiment 1 and 2 cloze tests explain the discrepancy in the outcomes of the experiments.
Table 18

Increase Between Experiments 1 and 2 in Error Rates and Proportions of Program-Dependent Cloze Items

<table>
<thead>
<tr>
<th>Program Version</th>
<th>Percent Increase Between Experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>Error Rate</td>
</tr>
<tr>
<td></td>
<td>262%</td>
</tr>
<tr>
<td>Hard</td>
<td>24%</td>
</tr>
</tbody>
</table>

Experimental Support for the Explanation of the Difference in the Outcomes of Experiments 1 and 2

To test the hypothesis that different every-nth deletion patterns with varying proportions of program-dependent cloze items may lead to different cloze test outcomes, Experiments 3 and 4 were conducted. The goal of the experiments was to conceptually replicate the outcomes of Experiments 1 and 2. As in the two previous experiments, the cloze tests used in Experiments 3 and 4 were constructed from two versions of a program. In Experiment 3 the cloze tests were designed to produce experimental differences. On the other hand, in Experiment 4 the cloze tests were designed to produce no experimental differences although the cloze tests were constructed from the identical programs used in Experiment 3.

The difference between the previous experiments and the proposed experiments is that, while in the previous experiments the proportion of program-dependent cloze items in each cloze test varied randomly, in the proposed experiments the proportion of program-dependent cloze items in each cloze test would be controlled. Instead of blindly selecting the cloze deletion patterns as in Experiments 1 and 2, it was decided to generate all the three possible 1-in-3 cloze tests for each of the Experiment 3 and 4 programs, calculate the proportion of program-dependent cloze items for each cloze test, and then use in each experiment two of the cloze tests hypothesized to have the proper proportions of program-dependent cloze items to replicate the effects found in Experiments 1 and 2. It should be noted that at the outset of
Experiments 1 and 2 it was assumed that differences in subject responses to the cloze tests would be indicative of differences in the programs being measured (as in the multiple-choice pilot studies). The assumption for Experiments 3 and 4, however, was that differences in subject responses were mainly indicative of cloze test differences and not program differences. The assumptions for Experiments 1 and 2 could likewise be revised to account for the influence of cloze test differences.

To replicate Experiment 1, it was planned that in Experiment 3 the cloze test selected for one program versions would have a larger proportion of program-dependent cloze items and the cloze test selected for the other program version would have a smaller proportion of program-dependent cloze items. The hypothesis of Experiment 3 would then be that subjects working with the cloze test with the larger proportion of program-dependent cloze items would have lower scores and would take longer to complete the test than subjects working with the cloze test with the smaller proportion of program-dependent cloze items. Correspondingly, to replicate Experiment 2, it was planned that in Experiment 4 the selected cloze tests would have similar proportions of program-dependent cloze items. The hypothesis of Experiment 4 would be that there would be no significant difference in the scores or completion times for the two cloze tests.

The two program used in Experiments 3 and 4 were both variations of the conventional version of the master file update program used in the previous experiments and pilot studies. One program (Appendix EE), termed the "parameters" version of the master file update program, was constructed from the conventional version of the program by changing all occurrences of multiple procedure entry points and associated code into separate procedures. For example, if the conventional version of the master file update program had a procedure with two entry points, a "main" entry point (indicated in PL/I by a PROCEDURE statement) and a "secondary" entry point (indicated in PL/I by an ENTRY statement), then the procedure was divided into two procedures. One new procedure
contained the code associated with the main entry point and the other new procedure contained the code associated with the secondary entry point.

The other program used in Experiments 3 and 4 [Appendix FF], known as the "globals" version of the master file update program, was constructed in the same manner as the parameters version except that, as well as eliminating all multiple entry points, all parameters and local variables were also eliminated. Instead of parameters and local variables, the globals program contained only global variables.

As constructed, the globals and parameters versions of the master file update program differed in only two respects: (1) the globals version of the program contained global variables, but no parameter lists or local variables, while the parameters version contained parameter lists and local variables, but no global variables and (2) the globals version contained variable declaration statements in only the main procedure while the parameters version contained variable declaration statements in every procedure. In all other respects the two programs were identical.

In Experiments 1 and 2 the only dependent variable was score. A second index of comprehension, time for completion, could not be used as a dependent variable (unlike in the two pilot studies) because of the different number of cloze items that had to be completed by the different treatment groups. Part of the rationale behind the design of the globals and parameters program versions was that their similarity would allow, using the proper cloze procedure, the construction of cloze tests with the same number of items. This meant that an improvement could be made in the experimental design of the previous cloze experiments by allowing time for completion as well as score to be used as dependent variables in Experiments 3 and 4.

To get the same number of items on cloze tests constructed from the globals and parameters programs, the cloze procedure from Experiment 2 had to be slightly modified for Experiments 3 and 4. The current cloze procedure not only made no deletions in variable declaration statements
(as in Experiment 2), but also made no deletions in parameter lists. Given that the error rate for parameter list cloze items in the conventional program version in Experiments 1 and 2 was so low (less than 3%), it was felt that excluding parameter list cloze items would have little effect on the outcome of the current experiments. Since the only differences in the two program versions were in their variable declaration statements and parameter lists, the remaining program code from which deletions could be made in the two programs was the same for both programs. Cloze tests made from the two programs using the same rate of deletion, therefore, had the same number of tokens. Furthermore, if the deletions began with the same starting token, the cloze items deleted from the globals program were not only the same in number, but identical to the items deleted from the parameters program.

Another part of the rationale behind the design of the globals and parameters program versions was that it was believed that cloze tests constructed from the two programs would exhibit a wide range of proportions of program-independent cloze items. Experience with categorizing cloze items had demonstrated to the author that variable declaration statements and parameter lists played a large role in causing cloze items to be program-independent. That is, information contained in variable declaration statements and parameter lists often can be used to successfully complete a cloze item without reference to the functionality of the program in which the cloze item is embedded. Since the parameters program had been designed with parameter lists and many variable declaration statements, the opposite of the design of the globals program, it was felt that cloze tests constructed from the parameters program should have larger proportions of program-independent cloze items than cloze tests constructed from the globals program. Also, if the assumption that the characteristics of every-nth cloze tests are not stable was valid, then it was felt that different every-nth cloze tests constructed from any one program should exhibit a range in the proportion of program-independent cloze items. This could make it possible to find cloze tests that satisfy the requirements of Experiment 3 (one cloze test with a larger proportion of program-
dependent cloze items than the other) as well as Experiment 4 (two cloze tests with the same proportion of program-dependent cloze items). This is in fact what occurred. The three cloze tests that could be constructed from the globals program version using a 1-in-3 deletion rate had, respectively, 32, 30, and 23 cloze items in the program-dependent category. The three possible cloze tests constructed from the parameters program using a 1-in-3 deletion rate had, respectively, 20, 19, and 17 cloze items in the program-dependent category. To achieve the maximum difference in the number of program-dependent cloze items, the globals program cloze test with 32 program-dependent cloze items and the parameters program cloze test with 17 program-dependent cloze items were used in Experiment 3. Likewise, to achieve the minimum difference in the number of program-dependent cloze items, the globals cloze test with 23 program-dependent cloze items and the parameters cloze test with 20 cloze items were used in Experiment 4.

**Experiment 3**

**Design and Overview.** A single-factor experiment was performed with the independent variable being program version (parameters and globals) and the dependent variables being the score and the time for completion on cloze tests. Since the number of items on the cloze tests for the different treatment groups was the same, score was defined as the number of items correctly completed on a test. A hypothesis of the experiment was that the cloze test results would indicate that the cloze test constructed from the parameters program version was easier than the cloze test constructed from the globals program version. In particular, the average score on the parameters program cloze test would be significantly higher than the average score on the globals program cloze test and that the average time to complete the parameters test would be significantly less than the average time to complete the globals test. As with the other cloze test experiments, an additional hypothesis was that the statistical results would not vary between a verbatim or synonymic scoring technique. Furthermore, it was also hypothesized that the error rate of the program-independent cloze items on the cloze tests
would be lower than the error rate for the corresponding program-
dependent items.

Subjects. Twenty of forty available students in two sessions of an
termediate level computer science course at The Ohio State University
participated in the experiment. Both sessions of the course had the
same instructor. Half of the subjects were in one session of the course
and half were in the other session of the course. All but one subject
were undergraduates. The remaining twenty subjects were reserved for
another experiment.

Materials. The experimental materials consisted of a background survey
and an experimental booklet. The background survey was the same as used
in the previous experiments and pilot studies [Appendix A]. The
experimental booklet contained an instruction sheet, a program
description, and a cloze test. The instruction sheet was the same as
the instruction sheet used in the previous cloze experiments [Appendix
T] except that the subjects were informed that they had approximately
forty-five minutes for the experiment (instead of the hour and forty-
five minutes for the previous cloze experiments). The program
description was also the same as the one used in the previous cloze
experiments [Appendix U] except the note discussing ENTRY statements and
STATIC variables was deleted since the program versions used in the
current experiments did not use ENTRY statements or STATIC variables.
The cloze test in the experimental booklet was either constructed from
the parameters or globals program version, which were coded in PL/I
[Appendices EE and FF, respectively]. Both forms of the cloze test were
formed using a 1-in-3 deletion pattern. To get the proper proportions
of program-dependent cloze items, the deletions for the parameters cloze
test [Appendix GG] began with the third token and the deletions for the
globals cloze test [Appendix HH] began with the first token. The
globals cloze test had 32 program-dependent cloze items while the
parameters cloze test had 17 program-dependent cloze items. Appendix DD
lists the items on the two cloze tests that were categorized as program-
dependent (items not listed were program-independent). Both cloze tests
had the same total number of items.
Procedure. The same general procedure was followed for the current and previous cloze experiments. The subjects were run as a group in a classroom setting. The subjects were balanced across treatment groups based on their PL/I programming experience as reported in the background survey completed before the experiment. Each subject was allowed forty-five minutes to complete the cloze test. There were few questions and the subjects did not express any difficulty in understanding the cloze task. Because of the lack of time, unlike the previous experiments and pilot studies, subjects were not asked to complete a post-experimental questionnaire at the end of the experiment. Subjects were allowed to leave the experiment once they had completed the cloze test.

Results. All but two subjects completed the experiment in the allotted time. The reliability of the cloze tests was analyzed using the Kuder-Richardson reliability estimate (KR-8). As with the previous cloze experiments, the reliability analysis was performed with cloze scores determined using the synonymic scoring technique. Inspection of Table 19 reveals that both cloze tests had high inter-item consistency.

Table 19

<table>
<thead>
<tr>
<th>Test (Program)</th>
<th>Kuder-Richardson Reliability Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>.90</td>
</tr>
<tr>
<td>Globals</td>
<td>.91</td>
</tr>
</tbody>
</table>

The score and time to completion results are reported in Table 20. One-tailed t-tests revealed significant score and time effects. Subjects working with the parameters program cloze test had significantly higher scores and significantly faster completion times than subjects working with the globals program cloze test. Table 20 also shows that the results hold for both the synonymic and verbatim scoring techniques. As in the previous cloze experiments, the synonymic and verbatim scores were highly correlated ($r = .99$, $p < .001$).
Table 20
Experiment 3 t-test Analysis of Score and Time to Completion

<table>
<thead>
<tr>
<th>Dependent Variable</th>
<th>Scoring Technique</th>
<th>Parameters</th>
<th>Globals</th>
<th>$t$</th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>Synonymic</td>
<td>M 50.5</td>
<td>41.9</td>
<td>2.71</td>
<td>.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SD 5.6</td>
<td>8.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Score</td>
<td>Verbatim</td>
<td>M 49.7</td>
<td>39.9</td>
<td>2.79</td>
<td>.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SD 6.2</td>
<td>9.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td>M 33.2</td>
<td>41.7</td>
<td>-3.01</td>
<td>.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SD 6.2</td>
<td>6.4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*a n = 10 for each group.

b Maximum score is 59, maximum time to completion is 49 minutes.

Table 21 shows that, as with Experiments 1 and 2, for both cloze tests the error rate for the program-independent cloze items was less than the overall error rate and that the error rate for the program-dependent cloze items was greater than the overall error rate. Note that the apparently large difference in the program-independent error rates for the parameters and globals programs was not statistically significant as revealed by a two-tailed $t$-test (there was no a priori reason to expect that one error rate would be larger or smaller than the other) ($t = -1.70$, n.s.). A one-tailed test was significant at the .05 level.

Table 21
Difficulty of Different Categories of Cloze Items in Experiment 3

<table>
<thead>
<tr>
<th>Cloze Item Category</th>
<th>Program Version</th>
<th>Program-Dependent</th>
<th>Program-Dependent</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>8.3</td>
<td>29.4</td>
<td>14.4</td>
<td></td>
</tr>
<tr>
<td>Globals</td>
<td>19.3</td>
<td>37.2</td>
<td>29.0</td>
<td></td>
</tr>
</tbody>
</table>
Responses to the background survey were correlated with the independent and dependent variables. As with the previous cloze experiments, the correlations were only performed with the synonymic scores because of the similarity between the scores as determined by the two scoring techniques. The means and standard deviations of the responses are reported in Appendix II and the correlations are reported in Appendix JJ. As usual, only significant correlations that hold for each treatment group as well as the combined groups will be reported to avoid spurious correlations.

There was no significant correlation between the number of PL/I programs written by a subject and the treatment group to which the subject was assigned. Also, the mean number of PL/I programs written by each group was similar (12.4 for the parameters group and 11.9 for the globals group). These results are interpreted to indicate that subjects were properly balanced across treatment conditions according to the balancing criterion. There were no background survey variables that were significantly correlated with a dependent variable for both the individual treatment groups and the combined groups.

Discussion. Experiment 3 was designed to replicate the results of Experiment 1. Two highly comparable program versions were developed for Experiment 3. One program, the "globals" version, solved the master file update problem using global variables, no parameter lists, and a minimum of variable declaration statements. The second program, the "parameters" version, solved the master file update problem using parameter lists, no global variables, and many more declaration statements than used by the globals version. The code of the two programs was identical except in variable declaration statements and parameter lists. Using a 1-in-3 deletion pattern, the three possible cloze tests for the globals program and the three possible cloze tests for the parameters program were constructed. The three cloze tests for each program were shown to exhibit a range of proportions of program-dependent cloze items. This fact supported the hypothesis that different every-nth cloze tests may have different characteristics and, thus, may not form a stable measure. Consistent with the
characteristics of the cloze tests used in Experiment 1, one cloze test used in Experiment 3 contained almost twice the proportion of program-dependent cloze items as the other cloze test. An improvement over Experiment 1 was that the two cloze tests used in Experiment 3 contained the same number of items and, therefore, time to complete the tests as well as score could be used as dependent variables in Experiment 3. Consistent with the results of Experiment 1 and as hypothesized, subjects working with the cloze test with the higher proportion of program-dependent cloze items took significantly longer to complete the test and scored significantly lower than the subjects working with the cloze test with the lower proportion of program-dependent cloze items. Also consistent with the results of Experiment 1 and as hypothesized, for both cloze tests in Experiment 3, the error rate of the program-dependent cloze items was at least twice the error rate of the other cloze items. Furthermore, Experiment 3 was consistent with the previous cloze experiments in that the cloze tests were shown to be highly reliable, the cloze scores were high (around 78%) compared to prose domain cloze scores (around 50%), and the score effects (or lack of effects) were the same using either the verbatim or synonymic scoring techniques.

The outcome of Experiment 3 may not be due only to cloze test differences, but also to program differences since the different treatment groups worked with different programs as well as cloze tests. In other words, it may be that the experimental outcome was not a result of features of the cloze tests, as hypothesized, but was a result of features of the programs. As it happens, the experimental outcome of Experiment 3 is consistent with the commonly accepted notion that global variables, like goto statements, are "harmful" and that the information hiding and controlled module interface features of parameter lists are "helpful." Thus, it may be natural to assume that subjects working with the globals program cloze test would have a more difficult experience as demonstrated by the score and time effects than subjects working with the parameters program cloze test. Because of the possibility that the outcome of Experiment 3 may be explained by program treatment...
differences and not necessarily cloze test treatment differences, another experiment was conducted. It was hoped that the next experiment would still support the hypotheses of Experiments 1, 2, and 3 concerning the influence of program-dependent cloze items while countering the notion that the effects of Experiment 3 were due to program treatment differences.

**Experiment 4**

The purpose of Experiment 4 was to give additional support to the hypothesis that the outcomes of Experiments 1 through 3 were due to the effects of differing proportions of program-dependent cloze items and to show that the results of Experiment 3 are not satisfactorily explained by the idea that the globals program is inherently more difficult to comprehend than the parameters program. Specifically, the goal was to replicate the outcome of Experiment 2 by showing that cloze tests constructed from the globals and parameters programs of Experiment 3 and with similar proportions of program-dependent items would not result in significant score or time effects. Since the only factor changed between Experiments 3 and 4 was the relative proportions of program-dependent cloze items on the cloze test, the hypothesized discrepancy in the outcomes of the two experiments should not be due to properties of the programs, which stayed fixed.

**Design and Overview.** The design of Experiment 4 was the same as the design of Experiment 3. A single-factor experiment was performed with the independent variable being program version (parameters and globals) and the dependent variables being the score and the time for completion on cloze tests. As with Experiment 3, the number of items on the cloze tests for the different treatment groups was the same and, therefore, score was defined as the number of items correctly completed on a test. Unlike Experiment 3, the main hypothesis of Experiment 4 was that the cloze test results would support the notion that the cloze tests constructed from the parameters and globals program versions were of the same difficulty. In particular, the average score on the parameters program cloze test would not be significantly different from the average
score on the globals program cloze test and that the average time to complete the parameters test would not be significantly different from the average time to complete the globals test. As with the other cloze test experiments, an additional hypothesis was that the statistical results would hold using either a verbatim or a synonymic scoring technique. Again consistent with the previous cloze experiments, it was also hypothesized that the error rate of the program-independent cloze items on the cloze tests would be lower than the error rate for the corresponding program-dependent items.

Subjects. The twenty remaining subjects not used in Experiment 3 participated in Experiment 4. The subjects were in two sections of an intermediate level computer science course at The Ohio State University. Both sections of the course had the same instructor. Half of the subjects were in one section of the course and half were in the other section of the course. All but three of the subjects were undergraduates.

Materials. The experimental materials were identical to those used in Experiment 3, except that the deletions in the globals program cloze test began with the third token instead of the first and the deletions in the parameters program cloze test began with the first token instead of the third. Appendix KK contains a listing of the parameters cloze test and Appendix LL contains a listing of the globals cloze test. Both cloze tests were written in PL/I. The experimental instruction sheet was the same as used in Experiment 3, which was the same as used in Experiments 1 and 2 (Appendix T) except that the subjects were informed that they had approximately forty-five minutes for the experiment (instead of the hour and forty-five minutes for Experiments 1 and 2). The program description was also the same as the one used in Experiment 3, which was also the same as the program description used in Experiments 1 and 2 (Appendix U) except that the note discussing ENTRY statements and STATIC variables was deleted since the program versions used in the current experiments did not use ENTRY statements or STATIC variables. As in Experiment 3, both of the cloze tests in the current
100

Experiment were formed using a 1-in-3 deletion pattern. The cloze tests selected for the experiment were such that the number of program-dependent cloze items on each test was similar (20 for the parameters cloze test and 23 for the globals). Appendix DD lists the items on the two cloze tests that were categorized as program-dependent (items not listed were program-independent). As in Experiment 3, both cloze tests had the same total number of items.

Procedure. The same general procedure was followed for the current and previous cloze experiments. The subjects were balanced across treatment groups based on their PL/I programming experience as reported in the background survey completed before the experiment and were run as a group and in a classroom setting. Each subject was allowed forty-five minutes to complete the cloze test. There were few questions and the subjects did not express any difficulty in understanding the cloze task. As with Experiment 3, there was insufficient experimental time for the subjects to complete the usual post-experimental questionnaire at the end of the experiment. Subjects were allowed to leave the experiment once they had completed the cloze test.

Results. All but one subject completed the experiment in the allotted time. The reliability of the cloze tests was analyzed using the Kuder-Richardson reliability estimate (KR-8). As with the previous cloze experiments, the reliability analysis was performed with cloze scores determined using the synonymic scoring technique. Inspection of Table 22 reveals that both cloze tests had high inter-item consistency.

Table 22
Experiment 4 Reliability Analysis

<table>
<thead>
<tr>
<th>Test (Program)</th>
<th>Kuder-Richardson Reliability Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>.92</td>
</tr>
<tr>
<td>Globals</td>
<td>.92</td>
</tr>
</tbody>
</table>
The score and completion time results are reported in Table 23. One-tailed t-tests did not reveal any significant effects. The score results are the same for both the synonymic and verbatim scoring techniques. As with the previous cloze experiments, the synonymic and verbatim scores were highly correlated (r = .97, p < .001).

Table 23

Experiment 4 t-test Analysis of Score and Time to Completion

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Scoring Technique</th>
<th>Parameters</th>
<th>Globals</th>
<th>t</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score Synonymic</td>
<td>M 46.8</td>
<td>48.4</td>
<td>-.45</td>
<td>.66</td>
<td></td>
</tr>
<tr>
<td>Score Verbatim</td>
<td>M 43.8</td>
<td>47.3</td>
<td>-.97</td>
<td>.35</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>M 39.0</td>
<td>37.6</td>
<td>.43</td>
<td>.67</td>
<td></td>
</tr>
</tbody>
</table>

* n = 10 for each group.

Maximum score is 59, maximum time to completion is 49 minutes.

Inspection of Table 24 shows that, as with all three of the previous cloze experiments, the error rate for the program-independent cloze items was less than the overall error rate and that the error rate for the program-dependent cloze items was greater than the overall error rate for both cloze tests.

Table 24

Difficulty of Different Categories of Cloze Items in Experiment 4

<table>
<thead>
<tr>
<th>Cloze Item Category</th>
<th>Program Version</th>
<th>Program-Independent</th>
<th>Program-Dependent</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>14.6</td>
<td>32.5</td>
<td>20.7</td>
<td></td>
</tr>
<tr>
<td>Globals</td>
<td>14.2</td>
<td>23.9</td>
<td>18.0</td>
<td></td>
</tr>
</tbody>
</table>
Responses to the background survey were correlated with the independent and dependent variables. As with the previous cloze experiments, the correlations were only performed with the synonymic scores because of the similarity between the scores as determined by the two scoring techniques. The means and standard deviations of the responses are reported in Appendix MM and the correlations are reported in Appendix NN. As usual, only significant correlations that hold for each treatment group as well as the combined groups will be reported to avoid spurious correlations.

As with all other experiments and pilot studies, there was no significant correlation between the number of programs written by a subject in the experimental materials implementation language (PL/I, in this case) and the treatment group to which the subject was assigned. Also, the mean number of PL/I programs written by each group was similar (11.6 for the parameters group and 9.9 for the globals group). These results are interpreted to indicate that subjects were properly balanced across treatment conditions according to the balancing criterion. There were no background survey variables that were significantly correlated with a dependent variable for both the individual treatment groups and the combined groups.

**Inter-Experimental Results**

As with Experiments 1 and 2, a comparison was made between the same cloze test used in Experiments 3 and 4. Unlike the cloze tests in Experiments 1 and 2, the only difference in the cloze tests used in Experiments 3 and 4 was the starting token for the deletion patterns. Table 25 shows that the error rate for the parameters cloze tests increased between the experiments and the error rate for the globals cloze tests decreased. The table also shows that the percentage of program-dependent items in the corresponding cloze tests increased or decreased in a similar manner.
To further examine the influence of program-dependent cloze items, a correlation was performed between the percentage of program-dependent items on a cloze test and a subject's corresponding percentage score for the eighty-four subjects used in Experiments 1 through 4. The correlation coefficient was calculated to be \(-.51\), which was highly significant \((p < .001)\). The coefficient indicated that as the percentage of program-dependent cloze items increased, subject scores tended to decrease. The regression equation relating a subject's percentage score \((y)\) to the percentage of program-dependent cloze items on the corresponding cloze test \((x)\) was \(y = -.55x + 103.4\). Table 25 may be used to compare the average percentage score for each cloze test used in Experiments 1 through 4 and the corresponding estimate of the average scores from the regression equation.

Table 25

<table>
<thead>
<tr>
<th>Percent Increase Between Experiments</th>
<th>Program Version</th>
<th>Error Rate</th>
<th>Proportion of Program-Dependent Cloze Items</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parameter</td>
<td>44%</td>
<td>18%</td>
</tr>
<tr>
<td></td>
<td>Global</td>
<td>-38%</td>
<td>-28%</td>
</tr>
</tbody>
</table>

Increase Between Experiments 3 and 4 in Error Rates and Proportions of Program-Dependent Cloze Items
Table 26
Comparison of Regression Estimates and Average Percentage Scores

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Cloze test (Program)</th>
<th>Regression Estimate</th>
<th>Average Percentage Score</th>
<th>Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Easy</td>
<td>96.8</td>
<td>96.5</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>88.6</td>
<td>88.5</td>
<td>.1</td>
</tr>
<tr>
<td></td>
<td>Easy</td>
<td>82.0</td>
<td>87.7</td>
<td>-5.7</td>
</tr>
<tr>
<td>2</td>
<td>Hard</td>
<td>82.5</td>
<td>85.7</td>
<td>-3.2</td>
</tr>
<tr>
<td></td>
<td>Parameter</td>
<td>87.4</td>
<td>85.6</td>
<td>1.8</td>
</tr>
<tr>
<td>3</td>
<td>Global</td>
<td>73.7</td>
<td>71.0</td>
<td>2.7</td>
</tr>
<tr>
<td>4</td>
<td>Parameter</td>
<td>84.7</td>
<td>79.3</td>
<td>5.4</td>
</tr>
<tr>
<td></td>
<td>Global</td>
<td>82.0</td>
<td>82.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

*Estimated percent score = -.55 * percent program-dependent items + 103.4, \( p < .001 \).

It is intriguing that if a cloze test is composed exclusively of program-dependent items, the regression equation predicts that the score on the test would be approximately 50%, a typical cloze score for the prose domain.

Experiment 3 allowed the influence of program-dependent cloze items to be examined in a different manner. Experiment 3 was the only cloze experiment with significant effects that used time to completion as a dependent variable. The correlation between time to completion and the program-dependent error rate for each subject was .45 (\( p < .05 \)) while the correlation between time to completion and the program-independent error rate was .33, which was not significant. This suggests that time to completion, like error rate, may be used as an indicator that program-dependent cloze items are more difficult than program-independent items.

As expected, there was a significant correlation across all four of the cloze experiments between the program-dependent and program-independent error rates for each subject (\( r = .60, \ p < .001 \)). This means that subjects that did poorly on the program-independent items tended to also
Discussion. The results of Experiment 4 gave additional support to the assumption that experimental outcomes are influenced by program-dependent cloze items. Using the same programs as in Experiment 3, but cloze tests with similar (instead of differing) proportions of program-dependent items, no significant score or completion time effects were found in Experiment 4. Just as with Experiments 1 and 2, if the assumption is made that the cloze tests used in the experiments are properly measuring program comprehension, then the outcomes of Experiments 3 and 4 are in conflict. Given that both Experiments 3 and 4 used subjects from the same population, that both experiments used similar materials and had similar experimental conditions, that subjects were balanced across all treatment conditions in the same manner, that all except three subjects completed the experiments within the allotted time, and that none of the subjects expressed any difficulty in the cloze task, it was assumed that, just as with Experiments 1 and 2, the difference in the outcomes of the experiments did not result from experimental design or procedure flaws. Thus, it is concluded that the cloze tests are not properly measuring program comprehension. Also, since the same programs were used in Experiments 3 and 4, the conflict in the outcomes of the experiments suggests that the results of the experiments are not explained by properties of parameters and globals, which did not change between the experiments. The only factor that did vary between Experiments 3 and 4 is the pattern of cloze items deleted from the programs used in the two experiments. In Experiment 3 (just as in Experiment 1), one cloze test had a larger proportion of program-dependent items than the other and significant score and completion time effects were found. In Experiment 4 (just as in Experiment 2), both cloze tests had close to the same proportion of program-dependent cloze items and no significant score or completion time effects were found. Also, just as with the previous cloze experiments, the cloze tests used in Experiment 4 were found to be highly reliable, the cloze scores were
high compared to prose domain cloze scores (about 81% vs. about 50%), and the score effects were independent of the scoring technique (synonymous or verbatim).

In all four of the cloze experiments conducted it has been shown that the error rate for program-dependent cloze items is greater than the error rate for program-independent cloze items. It has also been shown that, for all the experiments, the experimental results are consistent with a prediction based on the relationship in the proportions of program-dependent cloze items contained in the cloze tests used in the experiments. Across the experiments, the correlation between a subjects percentage score and the proportion of program-dependent cloze items in the corresponding cloze test was -.51, a high correlation coefficient for human-subject experimentation. In a post-experimental residual analysis it was also shown that a linear regression equation could be used to closely predict the percentage scores of the subjects used in the different experiment treatment groups based solely on the proportion of program-dependent items used in the corresponding cloze tests. These results are interpreted as giving strong support to the importance of program-dependent cloze items and the notion that every-nth cloze tests may not provide a stable measure of program comprehension. The significance of these results will be explored in the following chapter.
CHAPTER VI

Summary and Conclusions

Two main goals of software engineering are: (1) to decrease the cost of software and (2) to increase the reliability of software. Interestingly, these two goals parallel the two chief characteristics of "good" software engineering research methodology: (1) to be low in cost (to ensure wide-spread utility) and (2) to be valid and reliable (to ensure trustworthy results).

The close tie between software comprehensibility and software cost and reliability has led to many different techniques for improving software comprehensibility (e.g., top-down design, macro flowcharts, abstract data types, etc.). Reliable, science-based software development techniques along with the corresponding reliable, science-based research methodologies to uncover the techniques are obvious prerequisites to reliable software. Unfortunately, evidence for the efficacy and validity of the various techniques for improving software comprehensibility is more rooted in intuitive appeal, inspection, and everyday experience than in the more reliable knowledge gained from controlled experimentation. In other words, the support for many software development techniques tends to be more "experience-based" than "experiment-based", more practical than scientific.

Given the difficulties of performing controlled comprehension experimentation research, it is not surprising that there is more "soft" knowledge concerning software comprehension than "hard" knowledge. Most of these research difficulties are because comprehension research requires human-subject experimentation. Computer scientists are, in general, inexperienced and unknowledgeable in the area of human study. Also, the notorious unreliability of human subjects requires that statistical techniques and large groups of participants be used in human-subject studies to ensure trustworthy results. Also, the
specialized nature of the knowledge and skills often required of subjects for software engineering studies makes it difficult to locate qualified participants in large enough numbers.

Because human-subject study in computer science is still a largely unexplored field, new research methodologies and tools that are suited to exploration in this field still need to be developed. Of course, many tools and techniques can and should be borrowed from other research disciplines and then tailored to the needs of computer science. Although borrowing can aid the researcher, it means a software engineer must be a statistics-psychology-computer science interdisciplinarian, a rarity in this day of ever-greater specialization. A method for circumventing the interdisciplinary needs of the lone researcher is collaboration by experts within the different disciplines. However, the specialization of the experts also makes collaboration difficult.

Another, usually overriding difficulty in conducting human-subject study is caused by budget constraints. Even "small" studies can require many hours from many experienced (and thus, well-paid) computer science subjects and the use of complicated and expensive experimental conditions. Fruits from human-subject studies are often not reaped until after a long series of experiments has been replicated by several different researchers. Budgets, on the other hand, often encourage short-range thinking and the desire for immediate benefits. This tyranny of the budget holds even though it is well understood that the benefits from sacrificing short-term gains often far outweigh the costs of long-term planning. Excluding the metaphysical, good science is the only known technique for producing reliable knowledge and reliable knowledge, although difficult to achieve, ultimately leads one to the most practical and cost-effective path for solving a problem. Still, long-term scientific gains are often sacrificed for short-term "practical" gains.

Barry Boehm combined the notions of the need for the practical and the need for the scientific in his "classic" definition of software engineering:
Cloze testing is a comprehension measurement technique that seems to combine practical ease and theoretical soundness. Unfortunately, most of the research on cloze testing has been performed in the prose domain and little is yet known about cloze testing as a comprehension measurement technique in the software domain. The potential benefits of software cloze testing and the positive results from the few small-scale software cloze testing research efforts conducted to date suggest that greater gains might be had through more research.

On the practical side, it has been demonstrated in the prose domain that cloze tests have low development costs and that they are easy to administer, score, and interpret. On the other hand, multiple-choice quizzes, probably the most prevalent tool currently used for measuring software comprehension, can be difficult, and therefore expensive, to develop.

Research in the prose domain has also shown that cloze tests possess many positive scientific and theoretical benefits. Chief among these is that cloze tests have been shown to be a highly reliable and valid measure of prose comprehension. Also, cloze testing can be standardized, which has been a boon to comprehension research in the prose domain. Unlike cloze tests, multiple-choice comprehension quizzes cannot be standardized since the content of test questions is dependent on the test material. The lack of standardization makes it difficult to make general statements about the reliability and validity of multiple-choice quizzes.

Many researchers have been attracted to cloze testing because of its roots in information theory. In the language of information theory, a piece of text (e.g., a prose segment or a program) can be viewed as a "message" from a "sender" (the author of the text) to a "receiver" (the reader of the text). Just like in reading unmutilated text, a receiver
taking a cloze test attends to "redundancy" in the text (multiple sources of information provided by syntax and semantics) to make up for information losses caused by "noise" in the transmission of the message (cloze items).

Cloze tests seem to be "intrinsic measures of the effectiveness of communication by sampling the degree of language correspondence between a message source and a receiver" [RANK78, p. 151]. That is, cloze tests seem capable of measuring the inherent nature of comprehensibility by directly determining how close what an author "says" is to what a reader expects to "hear" and how well the reader can infer parts of the message. Inference is a basic component of all communication and comprehension. By interfering with the communication process itself, cloze tests seem to be a more direct measure of comprehension than other measures, such as multiple-choice quizzes, which treat comprehension as an after the fact "product" instead of as a "process" [RANK78].

Also of theoretical importance is that cloze tests seem to be a less biased metric than other comprehension measures. Measurement points within a piece of text are, to one degree or another, randomly selected when constructing a cloze test. The degree of individual control allowed in the construction of other types of comprehension measures makes it easy to inadvertently make the measure too easy or hard or biased in some other manner. Of course, the individual control offered by other measures can be of benefit, if properly handled.

Summary of Current Research

The initial intent of this research was to accumulate additional evidence of the validity and reliability of cloze testing as a software comprehensibility measure. The plan of attack was to demonstrate that cloze tests could determine (at a lesser cost in effort) the relative comprehensibility of two versions of a program whose comprehensibility had previously been established using multiple-choice comprehension quizzes. The first experiment was a success, but the results from a follow-up experiment disagreed with the results of the first experiment.
It was speculated that sample biasing caused by the every-nth selection technique had inadvertently confounded the measurement of the comprehensibility of the programs in the two experiments. A model of cloze tests was developed that suggests that cloze tests consist of easy-to-complete program-independent items and harder-to-complete program-dependent items. This model was used to resolve the conflict between the results of the first two experiments.

The model was also used to guide the selection of cloze tests that were used in two additional experiments. For these experiments it was hypothesized that the characteristics of the selected cloze tests, instead of the programs, could be used to predict the outcomes of the experiments. In the first of these experiments the model was used to select two cloze tests that were consistent with the hypothesis that one test, constructed from one version of a program, was more difficult to complete than the other cloze test, which had been constructed from another version of the program. The outcome of the experiment supported the hypothesis.

Finally, the model was used to guide the selection of two more cloze tests. Although the cloze tests were constructed from the same program versions used in the previous experiment, this time the model predicted that the characteristics of the two tests were such that the tests were of the same difficulty. Again, the outcome of the experiment supported the hypothesis.

**Experimental Conclusions**

The major topic of interest concerning software cloze testing is its reliability and validity. Since six out of the eight cloze tests used in the previous experiments had high Kuder-Richardson reliability estimates, we conclude that software cloze tests are reliable (as measured by inter-item consistency). The two cloze tests that had low (yet acceptable) reliability estimates may have been effected by ceiling effects and the chance occurrence of many items negatively correlated with total score. The validity of software cloze testing as indicated by the experiments, however, is more difficult to assess.
Based on the experiments performed, the types of validity appropriate for discussion are content validity, face validity, and criterion-related validity (no testing of construct validity was performed). As discussed in Chapter I, content validity refers to the degree to which a measure represents the breadth of the behavior domain being measured. In a technical sense, a cloze measure does a good job of "covering" a subject since it samples from all points of interest in the software. In contrast, it is difficult for a multiple-choice comprehension quiz to do as complete a job of covering a topic because of the usually small number of questions possible in the measure.

Software comprehension being a complex phenomenon that occurs at different "levels", such as the syntactic level and the semantic level, presents another dimension of content validity. In this regard, the nature of cloze testing is such that the correct completion of any cloze item is always minimally indicative of the syntactic level of comprehension (discounting, of course, completion by guessing). It has also been previously argued that at least some cloze items (the program-dependent ones) seem to require a certain degree of semantic comprehension to be correctly completed. The above arguments, combined with the strong theoretical relationship between the cloze testing process and comprehension previously presented suggest that software cloze tests have good content validity.

To check for face validity (whether a measure appears valid to an untrained observer), written comments were solicited from all the subjects in Experiments 1 and 2. Oral comments were also solicited during an informal post-experimental discussion from a group of subjects (graduate computer science students) that participated in an unreported cloze pilot study. The subjects were informed of the researcher's desire to use the cloze measure as a comprehension metric and were asked about what kinds of thought processes they used to complete the cloze tests and how they would go about testing the measure. Given that no subject expressed any skepticism concerning the cloze testing procedure, it is concluded that software cloze tests have good face validity.
The major difficulty with the validity of software cloze testing occurs when considering the outcome of the experiments and criterion-related validity. Criterion-related validity is concerned with how well a proposed measure of a given phenomenon corresponds to an accepted measure of the same phenomenon. The criterion measure used for the current experiments was a multiple-choice comprehension quiz. Since some of the cloze test results contradicted the multiple-choice quiz results, the validity of some cloze measures is brought into question. However, based on four arguments presented below, it is concluded that the research results do not support the notion that cloze testing is an inappropriate comprehension measure and, instead, raise a cautionary flag and indicate that the cloze procedure needs further tailoring to the software domain.

The first (and most important) argument is that analysis of the data seems to indicate that the experimental outcomes were caused by either unintentional (Experiments 1 and 2) or intentional (Experiments 3 and 4) differences in cloze sample characteristics and not by some organic flaw within software cloze testing. Since techniques may exist for either reducing cloze item sample differences or mitigating their effects, further research should be conducted. Cloze testing has the potential for a being a valid software comprehension measure, but some methodological improvements will first have to be made.

It has previously been suggested that a possible reason for differences in cloze sample characteristics is that the every-nth deletion technique is not stable. That is, different every-nth cloze tests generated from the same source material may not have the same characteristics because of differences in the cloze items sampled. This notion was suggested to the author by a study conducted by the prose researchers Meredith and Vaughan [MERE78] who argued that the properties of every-nth deletion patterns were potentially unstable and needed more research. As mentioned in Chapter II, other prose researchers have reached similar conclusions [see BOYCB2, ENTI78, PORT78, BORM64b]. Entin concluded from her software cloze testing research that "various deletion systems need
to be tried to see which one yields the most reliable and valid results." [ENTIBA, p. 46].

The Meredith and Vaughan study compared the properties of every-nth and random-deletion cloze patterns. Using the same text, the researchers constructed five 1-in-5 cloze tests (all that are possible) and five random-deletion cloze tests with twenty percent of the text randomly deleted. Each of ten groups of subjects worked with a different one of the ten different cloze tests. Statistical analysis showed that there were significant differences among the mean scores for the five groups working with the 1-in-5 cloze tests, but no differences among the mean scores for the five groups working with the random-deletion cloze tests. Such an analysis may as well hold true for the software domain and switching from every-nth deletion patterns to random-deletion patterns may be one way of better tailoring cloze testing methodology to the software domain.

Another possibility for improving the cloze sampling process is to increase the proportion of program-dependent items in each sample. The current research suggests that program-dependent cloze items significantly influence the outcome of cloze tests, while, on the other hand, program-independent cloze items tend to be easily completed and nondiscriminating. Studies of error rates suggest that variable declaration statement elements, I/O format specification elements, parameter list elements, and assignment statement operators could be eliminated from cloze samples without harm to the measure. These items are, in general, easy-to-complete program-independent items. If the supposition is true that the completion of program-dependent cloze items tends to require a substantially greater understanding of the functionality of a program than does the completion of program-independent items, then sampling more from the program-dependent items should cause a cloze test to better represent the program factors that affect the comprehension of the program and, thus, improve the measure.

There is also another potential solution to the sampling differences problem. Several prose researchers mitigate the effects of differences
in sample characteristics by combining the results from more than one
cloze measure. Sampling more than once from the same material and
averaging the results of the samples reduces the effects of sample
differences. This procedure should also work in the software domain.

It should be noted that the problem with differences in sampling
characteristics is only of concern when comparing the results of several
cloze tests (as when comparing the comprehensibility of several
programs). Another standard use for comprehension tests is to compare
the comprehension of a single program by several subjects. Here only
one cloze test is used and the need to compare samples disappears. Of
course, one may still be concerned with improving a single cloze measure
by improving on sampling techniques.

It is interesting to note that the ranges in the number of program-
dependent cloze items for the three possible easy program cloze tests
and the three possible hard program cloze tests analyzed in Experiments
3 and 4 did not overlap. Although it was possible to find proportions
that were similar in each range and thus make the comprehension
difference between the two programs disappear, the proportions did not
suggest that it would be possible to reverse the comprehension
difference and make the easy program cloze test more difficult than the
hard program cloze test. This, again, suggests that cloze tests have
the potential for properly tapping into program differences that affect
comprehension. It also suggests that software deletion strategies need
to be "fine tuned" to reduce the spread in the ranges of program-
dependent items and thereby reduce the chances of sampling differences.
Of course this argument is predicated on the assumption that the
parameters program is, indeed, less difficult to comprehend than the
globals program (as hypothesized and suggested by the current wisdom).

The second argument for favoring further research of software cloze
testing is that no one experiment provides conclusive evidence for or
against any hypothesis and, instead, only contributes another datum in
the ongoing experimental process. Given the potential benefits of
software cloze testing compared to the other accepted software
comprehension measures and that only a little research has been performed on cloze testing in the software domain, further research seems warranted.

The third argument for further research is related to the second and that is that several other software researchers have already supplied positive evidence concerning software cloze testing. The current research by no means "invalidates" the previous research. However, as previously stated, it does raise a cautionary flag. For example, the same type of post-experimental analysis as performed in the current Experiments 1 and 2 was performed on the two cloze tests used in the Cook experiment (COOK82). Using a multiple-choice test in a pilot study, Cook demonstrated that one program was more difficult to comprehend than another. Cloze tests also demonstrated the same comprehensibility relationship. Since the programs from which the cloze tests were constructed were semantically and syntactically similar (they both performed a Shell sort), the characteristics of the items on the tests could be compared. A classification conducted by this researcher of the cloze test items showed that the test for the hard program contained 50% program-dependent cloze items while the test for the easy program contained only 33%. Thus, as in the current Experiments 1 through 4, the outcome of the Cook experiment agreed with the relative balance of the program-dependent items between the cloze tests. However, if Cook had begun the deletion pattern for the hard program cloze test with the second token instead of the first (as used in the experiment), then the hard program cloze test would have had only 32% program-dependent cloze items instead of 50%. Perhaps then, as in the current experiments, the original comprehension difference would not have been detected. The current research results and the above analysis of the Cook experiment should warn future software cloze researchers to be cautious of cloze item sample differences.

The fourth argument in favor of continuing software cloze testing research, similar to the third, is that there is much research in support of cloze testing in the prose domain. Cloze testing is a
valued, long-standing, and widely used tool within the prose domain. Because of the standard form of cloze testing, researchers within the prose domain have been able to build on their individual results by combining the results of different experiments and experimenters. The idiosyncratic nature of comprehension measurement by different researchers within the software domain tends to disallow this kind of compounding of knowledge. The similarities in the prose domain and the software domain make it seem reasonable to assume that cloze testing can be successfully transferred to the software domain.

Of course, differences in prose and software suggest that cloze testing probably cannot be transferred to the software domain unchanged. Compared to typical cloze scores in the prose domain based on a 1-in-5 deletion rate, the cloze test scores found in this research were high even when a 1-in-3 deletion rate was used*. High software cloze scores when using a 33% deletion rate supports the notion that the syntax and semantics of software is much more highly structured than prose and, therefore, more predictable. A deletion rate as high as 50% from at least certain parts of text may be both necessary and useful in the software domain.

Contributing to the high software cloze scores is that many software cloze items (usually program-independent items) are "giveaways", i.e., they have a low error rate. As previously argued, not sampling from program-independent items should make a cloze sample better represent the program factors that affect the comprehension of the program and, thus, improve the validity of the measure. Also, because of their low error rate, program-independent items are generally nondiscriminating items and not sampling from them should tend to increase the reliability of the measure.

Several conclusions based on the current research also may be made concerning practical aspects of software cloze testing. Compared to the

*Look [COOK82] found similar high scores using a 1-in-5 deletion rate. Entin [ENTE84] found lower scores with a 1-in-5 rate, but used only subjects in introductory level computer science courses whereas the other software studies used subjects that were at least at an intermediate level.
alternatives, cloze testing was found to be a simple methodology for measuring comprehension. The tests were easy to construct, administer, score, and interpret. As with most other cloze research, the easy-to-use verbatim scoring technique was found to work as well as the more intuitively appealing, but difficult-to-use synonymic scoring technique.

The entire cloze process seems to be ideally suited to being computerized. Standard deletion patterns are determined by only two parameters (random or systematic pattern and rate of deletion) and should be easily automated. With a few changes existing "pretty formatters" should be capable of putting mutilated text into a presentable test form. Computerized cloze test administration also would greatly speed-up the scoring process, eliminate interpretation problems caused by differences in the handwriting of subjects, and allow greater tracking of subject behavior during the test taking process. Example behaviors that may be tracked are the order of item completion, focus of attention, and the completion time for individual cloze items. The measurement of the completion time for individual cloze items would give cloze researchers a second dependent variable for measuring the effort required to complete individual cloze items. Item error rate is currently the only such dependent variable.

It is important to note that it is not concluded from the current research that the relative proportions of program-dependent items contained by cloze tests constructed from two programs may, in general, be used as a predictive measure of the relative difficulty of the cloze tests (or the relative comprehensibility of the programs). The relative proportions of program-dependent cloze items was considered significant in the current experiments (and the Cook experiment) only because the programs compared in the experiments were highly similar syntactically and semantically. It is certainly possible that a cloze test with a low proportion of program-dependent items may be more difficult to complete than a cloze test with a higher proportion of program-dependent items because the program-dependent items of the first test are more difficult than the program-dependent items of the second test. The difference in
the difficulty rates could stem from differences in the functionality or other characteristics of the two programs. For example, a program-dependent item from a program that performs an n-way tape sort may be more difficult than a program-dependent item from a program that does a master file update.

Summary of Contributions of Research

This research made both theoretical and practical contributions toward the goal of establishing the cloze metric as a viable software comprehension measure. The theoretical contributions of the research will be considered first.

An important theoretical characteristic of a metric is its validity. As previously discussed, several researchers have examined the validity of software cloze tests and have concluded that the standard software cloze procedure possesses criterion-related validity. Other researchers have used software cloze metrics in experiments without questioning the validity of the measures. This was the first known research effort to suggest that the software cloze testing procedure as commonly used by software engineers has potential problems with criterion-related validity. As stated in the previous section, this should not discourage further cloze testing use or research. It should act instead as a cautionary flag and suggest to other researchers that care should be used when interpreting the results of standard cloze tests and that steps should be taken to ensure that a cloze test properly samples the test material being measured. Similar suggestions have been made by several cloze researchers in the prose domain concerning prose-based cloze tests.

Besides suggesting the need for caution, this research also offers to other investigators a research model of software cloze items. The model suggests that software cloze tests simultaneously measure at least two levels of cognitive processing. The model has been shown to explain the criterion-related validity problem encountered in the current experiments and also has been shown to be applicable to the research of other software cloze testing investigators. The model suggests to other
researchers that better software cloze tests may be developed using a
guided deletion strategy. The model also suggests a category of cloze
items that may be better suited for revealing semantics-oriented
software comprehension and another category that may be of little use in
studying experienced subjects. Further methods suggested by the model
for improving software cloze testing research are detailed in the next
section.

This was also the first known research effort to investigate the
internal reliability of software cloze tests. Given that seven of the
original eight different cloze tests in this research had acceptable
reliability estimates (and that six of the estimates were high), future
researchers should be confident in the reliability of their software
cloze metrics. Based on the questions asked during the current
experiments and the comments solicited afterwards, future researchers
should also be confident of the face validity of the software cloze
metric. Highly experienced as well as less experienced computer science
students used in the current research expressed little trouble with the
software cloze testing procedure.

Beside the above theoretical contributions, this research has
demonstrated that time-to-completion may be used successfully as a
dependent variable in cloze experiments. This adds an important second
research tool to the software cloze testing researcher's toolbox (along
with error rate). It has also been demonstrated that an item deletion
rate higher than 20% may be used when constructing software cloze tests.
In fact, the current research suggest that a deletion rate greater than
20% may be useful to avoid score ceiling effects and to otherwise
improve the statistical characteristics of software cloze tests.

On the practical side, this research adds to the growing evidence that
software cloze tests are easy to develop, administer, and score.
Support is given also to the notion that the easier-to-use verbatim
scoring technique may be as satisfactory as the more intuitively
satisfying synonymic scoring technique.
Future Research Suggestions

As indicated by the current research, software cloze samples may not properly represent program aspects that affect comprehension. Future research efforts should explore the several methods previously mentioned for improving sample characteristics and mitigating the effects of differences in sample characteristics.

A software version of the Meredith and Vaughan study (MERE78) could be conducted to see if random deletion patterns offer an improvement over every-nth deletion patterns. The objection that random deletion patterns allow an unnecessarily large number of tokens between deletions could be removed by testing a combination of the random and every-nth deletion strategies whereby a single token within each group of N tokens is randomly deleted.

Future studies could also explore the effects of increasing the proportion of program-dependent items in a cloze sample. It has been argued that greater sampling of program-dependent cloze items should improve the measure since it is felt that completing these items is a greater indication of program comprehension than completing program-independent items. Of course, increasing the proportion of program-dependent items in a sample requires that the category of an item be determined. The method used in the current research to determine whether an item was program-dependent or program-independent was context sensitive, subjective, and difficult to apply. What is needed is an a priori classification of various types of tokens (format elements, assignment operators, etc.). Although it is true that the program-independent and program-dependent categories are context sensitive, experience has shown that it is rare for a given cloze item to switch categories from one deletion pattern to the next.

Two methods come to mind for objectively determining if a given cloze item, in general, is a program-independent item or program-dependent item. One method is to perform a protocol analysis and ask subjects what type of information they are using to complete items as they perform a cloze task. If the subjects indicate that they complete
certain items using techniques that do not involve an understanding of the functionality of the program (such as listed in Chapter V), then these items are program-independent for at least the given program. The other items are, of course, program-dependent. If a protocol analysis shows that the same types of items are categorized the same across many different programs, then it should be possible to make general statements about the proper category for the items.

A second method for categorizing cloze items is to construct a standard cloze test and then scramble the order of the statements in the test. Items that most subjects can complete in the scrambled test should be program-independent. Program-dependent items should have a lower completion rate since it should be difficult to decipher the functionality of the program from the scrambled statements. Again, this procedure would have to be performed with several different programs to allow for general conclusions about item categories.

As previously mentioned, a possibility for mitigating the effects of sample differences is to use multiple cloze tests. This procedure is recommended by Bormuth for the prose domain [BORM64b] and also may be effective in the software domain. Software studies should be conducted to compare the effects of single and multiple cloze tests to explore this possibility.

A suggestion for most all future software cloze research is to measure more complex software than has been measured to date. Although the software used in the current research was less toy-like than that used by other known software cloze researchers, it is felt that the performance of the measure would have been better if it had been applied to even more complex software. The supposition behind this suggestion is that increasing the complexity of software should increase the proportion of program-dependent items within the software and thereby improve the measure as previously argued.

The current research software may be used for an example of the relationship between software complexity and program-dependent cloze
items. The current software consisted of many short procedures and each procedure only contained a small number of variables. The limited number of variables used in each procedure meant that the candidate set of answers for missing-variable cloze items was small. Even if a missing variable could not be determined using only program-independent information, the small candidate set made it easy to guess.

By possibly increasing the proportion of program-dependent items on a test, the use of more complex software should also lower the high scores typical for software cloze tests. Ideally, the scores should be lowered to about 50%, which is typical for cloze scores in the prose domain. Statistically, a test with a difficulty rate of 50% has the greatest power of discrimination. The use of a 1-in-2 deletion rates should also help lower the scores.

Future researchers should also explore the construct validity of cloze testing as a comprehension measure. One method for doing this is to administer a single cloze test to several groups of subjects where each group represents a different level of software experience. If the cloze test program is sufficiently difficult to comprehend that it does not cause ceiling effects, but not so easy to comprehend that it causes floor effects, then it would be expected that the different groups would demonstrate a range of comprehension scores corresponding to their experience level.

Future researchers are encouraged to computerize their cloze experimentation. Not only may this benefit the individual experimenter as previously outlined, but it should also encourage the spread of the cloze measure to other researchers by increasing its ease of use.

Finally, independent of the utility of cloze testing for software, cloze testing may be a much needed tool for improving two important prose-like texts: software documentation and software specifications. Not only may cloze tests constructed from these texts indicate their readability (as with more common prose texts), but the actual study of the texts necessary to complete cloze tests may be a way of discovering inconsistencies and "holes" in the texts. In this sense cloze testing
is similar to error "seeding" for program testing purposes. As one attempts to remove the seeded errors (fill-in the cloze items), other errors may be revealed.

It is hoped that the pursuit of these ideas will help to better tailor the cloze testing process to the software domain and thereby encourage the adoption of cloze testing as a standard software engineering research tool.
BIBLIOGRAPHY


P. B. Moranda, "Software quality technology: (sad) status of; (unapproached) limits to; (manifold) alternatives to," Computer, vol. 11, no. 11, November 1978, pp. 72-78.


APPENDIX A

Background Survey for All Pilot Studies and Experiments

BACKGROUND SURVEY

NAME (please print): ______________________

Instructions: Please answer the questions on both sides of this sheet. Circumstances may require that some of your responses be approximations of or "best guesses" at the actual answers. Since the responses will be analyzed with a standard computer statistics package it is important that you answer all of the questions and that each answer is in the form of a single number.

___ A. What is your age?
___ B. What is your sex?
   1 = female
   2 = male
___ C. What is your class rank?
   1 = Freshman
   2 = Sophomore
   3 = Junior
   4 = Senior
   5 = Grad
___ D. What is your overall grade point average? (If you are a graduate student only consider your graduate GPA)
___ E. How many computer courses have you taken that involved programming?
___ F. What is your grade point average from these courses only?
   For questions that ask "How many years...?", there is no need for your answer to be more exact than to the nearest tenth of year.
___ G. How many years of education have you had since high school?
___ H. How many years of professional programming experience have you had?
___ I. How many years of experience have you had in teaching computer programming?
___ J. How many years have you been programming?
___ K. How many years have you been programming with Fortran?
___ L. How many years have you been programming with PL/I or PL/C?
___ M. How many years has it been since you have written a program?
N. How many years has it been since you have written a program in Fortran? (mark "X" if never)

O. How many years has it been since you have written a program in PL/I or PL/C? (mark "X" if never)

P. How many programs have you ever written?

Q. How many lines long was your longest program?

R. How many programs have you ever written in Fortran?

S. How many lines long was your longest Fortran program?

T. How many programs have you ever written in PL/I or PL/C?

U. How many lines long was your longest PL/I or PL/C program?

V. How many programming languages have you ever programmed in?

W. On a scale from 1 to 5 how do you rate your experience with WYLBR?

1 = no experience
5 = very experienced
APPENDIX B
General Instruction Sheet for Pilot Studies

DO NOT OPEN EXPERIMENT BOOKLETS UNTIL SO INSTRUCTED

Enclosed are an experiment booklet and a program booklet. When you are
told to start, please follow the instructions in the experiment booklet.

You will have approximately forty-five minutes to work with the
materials. Please work as quickly and accurately as possible. Turn in
the materials as soon as you are finished with them. You will then be
given a final questionnaire to fill in.

Note that not everyone is participating in the same experiment. You
should, therefore, not be concerned if some people finish before you do.
APPENDIX C

Program Description for Pilot Study 1

PROGRAM DESCRIPTION

Purpose: Update a master file
Input: 1. A master file (known as the old master file)
       2. A transaction file
Output: Another master file (known as the new master file)

Old master file description:

Each record consists of:
1. A bank account number (the key)
2. A bank balance
3. A name
4. An address

- Keys may range in value from 1 to 999
- No two records have the same key
- Records are in ascending key order

Transaction file description:

Each record consists of:
1. A transaction code
2. A bank account number (the key)
3. A transaction amount
4. A name
5. An address

- Keys may range in value from 1 to 999
- Records are in ascending key order. Records with the same key are in the order as they were created in real-time.

New master file description:

- Contains the same records as the old master file (and in the same format) except there may be some added, missing (deleted), or changed records as specified by transaction records.
- No two records have the same key
- Records are in ascending key order

Transactions code meanings:

<table>
<thead>
<tr>
<th>CODE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unless subsequent transactions specify otherwise, a record with the relevant information given in the transaction record is to be added to the new master file. Known as an &quot;add&quot; transaction.</td>
</tr>
<tr>
<td>2</td>
<td>Unless subsequent transactions specify otherwise, the information currently associated with the specified key is not to be included in the new master file. Known as a &quot;delete&quot; transaction.</td>
</tr>
</tbody>
</table>
If the name field of the transaction is not blank, then the name field of the new master file record currently associated with given key is to be changed to the indicated name. Similarly for the address field of the transaction record. Known as a "replace" transaction.

The balance field of the new master file record currently associated with the given key is to be decreased by the indicated amount. Known as a "debit" transaction.

The balance field of the new master file record currently associated with the given key is to be increased by the indicated amount. Known as a "credit" transaction.

Error processing:

There is no error detection - perfect input data is assumed.

Rational for program structure:

The program has been structured into several small routines to aid continued development. Routines that may be considered "too small" may be expected to grow as the problem specification grows in complexity.
APPENDIX D

Listing of Easy Version of Program for Pilot Study 1

10. C
20. C UPDATE FILE
30. C
40. C INTEGER TCODE, TKEY, OMKEY, NMKEY, CURKEY
50. C REAL TAMT, NMBAL, OMBAL
60. C CHARACTER*5 INUSE
70. C CHARACTER*35 TNAME, TADDR, NMNAME, NMADDR, OMNAME, OMADDR
80. C
90. C CALL INTRAN(TCODE, TKEY, TAMT, TNAME, TADDR)
100. C CALL INMAST(OMKEY, OMBAL, OMNAME, OMADDR)
110. C CALL NXTKEY(TKEY, OMKEY, CURKEY)
120. C DO WHILE (CURKEY .NE. 9999)
130. C CALL PKEY(TCODE, TKEY, TAMT, TNAME, TADDR, OMKEY, OMBAL, *
140. C * OMNAME, OMADDR, NMKEY, NMBAL, NMNAME, NMADDR, *
150. C END WHILE
160. STOP
170. END

180. C SUBROUTINE INTRAN(TCODE, TKEY, TAMT, TNAME, TADDR)
190. C
200. C READ A TRANSACTION RECORD
210. C
220. C INTEGER TCODE, TKEY
230. C REAL TAMT
240. C CHARACTER*35 TNAME, TADDR
250. C FORMAT(11,1X, I5, F5.2, A3, A3, A35)
260. C
270. C READ(5,10) TCODE, TKEY, TAMT, TNAME, TADDR
280. C AT END DO
290. C TKEY = 9999
300. C END AT END
310. C RETURN
320. C END

330. C SUBROUTINE INMAST(OMKEY, OMBAL, OMNAME, OMADDR)
340. C
350. C READ AN OLD MASTER RECORD
360. C
370. C INTEGER OMKEY
380. C REAL OMBAL
390. C CHARACTER*35 OMNAME, OMADDR
400. C FORMAT(13, F7.2, A35, A35)
410. C
420. C READ(19,10) OMKEY, OMBAL, OMNAME, OMADDR
430. C AT END DO
440. C OMKEY = 9999
450. C END AT END
460. C RETURN
470. C END

136
SUBROUTINE NXTKEY(TKEY, OMKEY, CURKEY)

C CHOOSE NEXT KEY

C INTEGER TKEY, OMKEY, CURKEY

IF (OMKEY .LE. TKEY) THEN
  CURKEY = OMKEY
ELSE
  CURKEY = TKEY
END IF

RETURN
END

SUBROUTINE PKEY(TCODE, TKEY, TAMT, TNAME, TADDR, OMKEY, OMBAL, NMKEY, NMBAL, NMNAME, NMADDR, INUSE, CURKEY)

C PROCESS ONE KEY

C INTEGER TCODE, TKEY, OMKEY, NMKEY, CURKEY

C REAL TAMT, NMBAL, OMBAL

C CHARACTER*35 INUSE

C CHARACTER*35 TNAME, TADDR, NMNAME, NMADDR, OMNAME, OMBADDR

C CALL INIT(NMKEY, NMBAL, NMNAME, NMADDR, OMKEY, OMBAL, OMNAME, OMBADDR, INUSE, CURKEY)

C WHILE (TKEY .EQ. CURKEY) DO

C CALL PTRANS(TCODE, TKEY, TAMT, TNAME, TADDR, NMKEY, NMBAL, NMNAME, NMBADDR, INUSE)

C END WHILE

C CALL FINAL(NMKEY, NMBAL, NMNAME, NMADDR, INUSE)

C CALL NXTKEY(TKEY, OMKEY, CURKEY)

RETURN
END

SUBROUTINE INIT(NMKEY, NMBAL, NMNAME, NMADDR, OMKEY, OMBAL, OMNAME, OMBADDR, INUSE, CURKEY)

C INITIAL STATUS

C INTEGER OMKEY, NMKEY, CURKEY

C REAL NMBAL, OMBAL

C CHARACTER*35 INUSE

C CHARACTER*35 NMNAME, NMADDR, OMNAME, OMBADDR

IF (OMKEY .EQ. CURKEY) THEN

NMKEY = OMKEY
NMBAL = OMBAL
NMNAME = OMNAME
NMADDR = OMBADDR
INUSE = 'TRUE'
END IF

CALL INMAST(OMKEY, OMBAL, OMNAME, OMBADDR)

ELSE

INUSE = 'FALSE'
END IF

RETURN
END
SUBROUTINE PTRANS(TCODE, TKEY, TAMT, TNAME, TADDR, NMKEY, NMBAL, NMNAME, NMADDR, INUSE)

PROCESS ONE TRANSACTION

INTEGER TCODE, TKEY, NMKEY
REAL TAMT, NMBAL
CHARACTER*5 INUSE
CHARACTER*35 TNAME, TADDR, NMNAME, NMADDR

CALL MODIFY(TCODE, TKEY, TAMT, TNAME, TADDR, NMKEY, NMBAL, NMNAME, NMADDR, INUSE)
CALL INTRAN(TCODE, TKEY, TAMT, TNAME, TADDR)
RETURN
END

SUBROUTINE FINAL(NMKEY, NMBAL, NMNAME, NMADDR, INUSE)

FINAL STATUS

INTEGER NMKEY
REAL NMBAL
CHARACTER*5 NMNAME, NMADDR

FORMAT(1X,13,1X,F7.2,1X,A35,1X,A35)

IF (INUSE .EQ. 'TRUE') THEN
WRITE(6,10) NMKEY, NMBAL, NMNAME, NMADDR
END IF
RETURN
END

SUBROUTINE MODIFY(TCODE, TKEY, TAMT, TNAME, TADDR, NMKEY, NMBAL, NMNAME, NMADDR, INUSE)

MODIFY NEW MASTER

INTEGER TCODE, TKEY, NMKEY
REAL TAMT, NMBAL
CHARACTER*5 INUSE
CHARACTER*35 TNAME, TADDR, NMNAME, NMADDR

IF (TCODE .EQ. 1) THEN
CALL PADD(TKEY, TAMT, TNAME, TADDR, NMKEY, NMBAL, NMNAME, NMADDR, INUSE)
ELSE IF (TCODE .EQ. 2) THEN
CALL PDDEL(INUSE)
ELSE IF (TCODE .EQ. 3) THEN
CALL PREPL(TNAME, TADDR, NMNAME, NMADDR)
ELSE IF (TCODE .EQ. 4) THEN
CALL PDEBIT(TAMT, NMBAL)
ELSE IF (TCODE .EQ. 5) THEN
CALL PCREDT(TAMT, NMBAL)
END IF
RETURN
END
139

1590.  SUBROUTINE PADD(TKEY,TAMT,TNAME,TADDR,NMKEY,NMBAL,NMNAME,
1600.                    NMADDR,INUSE)
1610.  
1620.  C  PROCESS DIFFERENT MODIFICATIONS
1630.  C  ADDITION
1640.  C
1650.  C    INTEGER TKEY,NMKEY
1660.  C    REAL TAMT,NMBAL
1670.  C    CHARACTER*5 INUSE
1680.  C    CHARACTER*35 TNAME,TADDR,NMNAME,NMADDR
1690.  C
1700.  C    NMKEY = TKEY
1710.  C    NMBAL = TAMT
1720.  C    NMNAME = TNAME
1730.  C    NMADDR = TADDR
1740.  C    INUSE = 'TRUE'
1750.  C    RETURN

1760.  C
1770.  C  ENTRY PDEL(INUSE)
1780.  C
1790.  C  DELETION
1800.  C
1810.  C    INUSE = 'FALSE'
1820.  C    RETURN

1830.  C
1840.  C  ENTRY PREPL(TNAME,TADDR,NMNAME,NMADDR)
1850.  C
1860.  C  REPLACEMENT
1870.  C
1880.  C    IF (TNAME .NE. ' ') THEN
1890.  C      NMNAME = TNAME
1900.  C    END IF
1910.  C    IF (TADDR .NE. ' ') THEN
1920.  C      NMADDR = TADDR
1930.  C    END IF
1940.  C    RETURN

1950.  C
1960.  C  ENTRY PDEBIT(TAMT,NMBAL)
1970.  C
1980.  C  DEBIT
1990.  C
2000.  C    NMBAL = NMBAL - TAMT
2010.  C    RETURN

2020.  C
2030.  C  ENTRY PCREDIT(TAMT,NMBAL)
2040.  C
2050.  C  CREDIT
2060.  C
2070.  C    NMBAL = NMBAL + TAMT
2080.  C    RETURN
2090.  C    END
APPENDIX E
Listing of Hard Version of Program for Pilot Study 1

10. C
20. C UPDATE FILE
30. C
40. INTEGER KEY, CODE
50. REAL AMOUNT
60. CHARACTER*5 ENDFLG
70. CHARACTER*35 NAME, ADDR
80. C
90. CALL BGINI
100. CALL BGOUT
110. CALL GET(KEY, AMOUNT, NAME, ADDR, ENDFLG)
120. WHILE (ENDFLG .EQ. 'FALSE') DO
130. CALL PREC(KEY, AMOUNT, NAME, ADDR, CODE, ENDFLG)
140. END WHILE
150. CALL ENDOUT
160. STOP
170. END

180. C SUBROUTINE PREC(KEY, AMOUNT, NAME, ADDR, CODE, ENDFLG)
190. 200. C PROCESS ONE RECORD
210. C
220. INTEGER KEY, CODE
230. REAL AMOUNT
240. CHARACTER*5 NEW, ENDFLG
250. CHARACTER*35 NAME, ADDR
260. C
270. CALL NEWGRP(NEW)
280. IF (NEW .EQ. 'TRUE') THEN
290. CALL NEWKEY
300. END IF
310. CALL PTYPE(CODE)
320. IF (CODE .EQ. 1) THEN
330. CALL PADD(KEY, AMOUNT, NAME, ADDR)
340. ELSE IF (CODE .EQ. 2) THEN
350. CALL PDEL
360. ELSE IF (CODE .EQ. 3) THEN
370. CALL PREPL(NAME, ADDR)
380. ELSE IF (CODE .EQ. 4) THEN
390. CALL PDEBIT(AMOUNT)
400. ELSE IF (CODE .EQ. 5) THEN
410. CALL PCREDIT(AMOUNT)
420. END IF
430. C
440. CALL GET(KEY, AMOUNT, NAME, ADDR, ENDFLG)
450. RETURN
460. END

140
SUBROUTINE BGNIN

INPUT MODULE

CALL INIT INPUT

INTEGER TCODE, TKEY, OMKEY, CURKEY, KEY, TYPE, CODE
REAL TAMT, OMBAL, AMOUNT
CHARACTER*5 NEWFLG, ENDFLG, NEW
CHARACTER*35 TNAME, TADDR, OMNAME, OMADDR, NAME, ADDR

FORMAT(I1, I1, I3, F5.2, A35, A35) 10
FORMAT(I3, F7.2, A35, A35) 20

READ(5,10) TCODE, TKEY, TAMT, TNAME, TADDR

AT END DO
TKEY = 9999
END AT END

READ(19,20) OMKEY, OMBAL, OMNAME, OMADDR

AT END DO
OMKEY = 9999
END AT END

CURKEY = 0

NEWFLG = 'FALSE'

RETURN

ENTRY GET(KEY, AMOUNT, NAME, ADDR, ENDFLG)

GET ONE RECORD

IF ((OMKEY .EQ. 9999) .AND. (TKEY .EQ. 9999)) THEN
ENDFLG = 'TRUE'
RETURN
END IF

ENDFLG = 'FALSE'

IF (OMKEY .LE. TKEY) THEN
KEY = OMKEY
AMOUNT = OMBAL
NAME = OMNAME
ADDR = OMADDR
TYPE = 1
READ(19,20) OMKEY, OMBAL, OMNAME, OMADDR

AT END DO
OMKEY = 9999
END AT END

ELSE
KEY = TKEY
AMOUNT = TAMT
NAME = TNAME
ADDR = TADDR
TYPE = TCODE
READ(5,10) TCODE, TKEY, TAMT, TNAME, TADDR

AT END DO
TKEY = 9999
END AT END
END IF

IF (KEY .EQ. CURKEY) THEN
NEWFLG = 'FALSE'
ELSE
NEWFLG = 'TRUE'
CURKEY = KEY
END IF

RETURN
1080. C ENTRY PTYPE(CODE)
1090. C GET PROCESSING TYPE
1100. C CODE = TYPE
1110. C RETURN
1120. C ENTRY NEWGRP(NEW)
1130. C GET GROUP STATUS
1140. C NEW = NEWFLG
1150. C RETURN
1160. C END
1170. C SUBROUTINE BGNOUT
1180. C OUTPUT MODULE
1190. C INITIALIZE OUTPUT
1200. C INTEGER KEY,NMKEY
1210. C REAL AMOUNT,NMBAL
1220. C CHARACTER*5 BUFFER
1230. C CHARACTER*35 NMNAME,NMADDR,NAMEN,ADDR
1240. C FORMAT(1X,13,1X,F7.2,1X,A35,1X,A35)
1250. C BUFFER = 'EMPTY'
1260. C RETURN
1270. C ENTRY PADD(KEY,AMOUNT,NAMEN,ADDR)
1280. C PROCESS ADDITION
1290. C NMKEY = KEY
1300. C NMBAL = AMOUNT
1310. C NMNAME = NAME
1320. C NMADDR = ADDR
1330. C BUFFER = 'FULL'
1340. C RETURN
1350. C ENTRY PDEL
1360. C PROCESS DELETION
1370. C BUFFER = 'EMPTY'
1380. C RETURN
1390. C ENTRY PREPL(NAME,ADDR)
1400. C PROCESS REPLACE
1410. C IF (NAME .NE. ' ') THEN
1420. C NMNAME = NAME
1430. C END IF
1440. C IF (ADDR .NE. ' ') THEN
1450. C NMADDR = ADDR
1460. C END IF
1470. C RETURN
1670. C ENTRY PDEBIT(AMOUNT)
1680. C PROCESS DEBIT
1690. C NMBAL = NMBAL - AMOUNT
1700. C RETURN
1710. C ENTRY PCREDIT(AMOUNT)
1720. C PROCESS CREDIT
1730. C NMBAL = NMBAL + AMOUNT
1740. C RETURN
1750. C ENTRY NEWKEY
1760. C PROCESS A NEW KEY
1770. C IF (BUFFER .EQ. 'FULL') THEN
1780. C WRITE(6,10) NMKEY,NMBAL,NMNAME,NMADDR
1790. C BUFFER = 'EMPTY'
1800. C END IF
1810. C RETURN
1820. C ENTRY ENDOUT
1830. C FINALIZE OUTPUT PROCESSING
1840. C IF (BUFFER .EQ. 'FULL') THEN
1850. C WRITE(6,10) NMKEY,NMBAL,NMNAME,NMADDR
1860. C END IF
1870. C RETURN
1880. C END
APPENDIX F

Quiz Instruction Sheet

The multiple choice questions on the following pages are concerned with the program in the accompanying booklet. For each of the questions, circle the number corresponding to the one best alternative answer. Be sure to consider all of the choices before making your decision. Feel free to write on any of these materials.

If you have any questions, raise your hand. Work as quickly and accurately as possible. Turn in these materials as soon as you are finished answering the questions.
APPENDIX G
Quiz for Easy Version of Program for Pilot Study 1

A. What might be caused by processing a transaction record with an invalid transaction code (TCODE)?

1. The incorrect transaction record will have no effect on the resulting new master file.
2. An infinite loop may be caused.
3. The resulting new master file may contain a record it would not otherwise contain.
4. The answer will vary depending upon factors not stated.
5. The incorrect transaction record will have no effect on the resulting new master file only if it is the last record in the transaction file.

B. What is the total number of assignment statements executed if the old master file contains only one record and the transaction file is empty?

1. 7
2. 21
3. 19
4. 9
5. none of the above

C. What is the maximum number of assignment statements that may be executed after OMKEY and TKEY both become equal to 9999?

1. 3
2. 7
3. 1
4. 11
5. none of the above
D. Assuming correct input data, what might be caused by deleting line number 1740?

1. The resulting new master file may contain keys found in the transaction file and not in the old master file and may contain keys found in the old master file and not in the transaction file.

2. The resulting new master file will only contain keys that are also found in the old master file.

3. The resulting new master file will always be empty.

4. The resulting new master file will only contain keys that are also found in the transaction file.

5. None of the above

E. Assume the old master file and the transaction file contain the indicated information. How many times is the subprogram PADD called?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #)</td>
<td>(record #)</td>
</tr>
<tr>
<td>(key)</td>
<td>(code)</td>
</tr>
<tr>
<td>(key)</td>
<td>(key)</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

1. 8
2. 2
3. 6
4. 5
5. 4

F. What will be caused by processing a "credit" transaction record with a key not matching any key in the old master file or the rest of the transaction file?

1. An undefined variable may be referenced.

2. A new master record might have its balance incorrectly increased.

3. A new master record might have an undefined balance.

4. 1 and 3 are correct

5. 1, 2, and 3 are correct
G. What might be caused by processing an "add" transaction record having the same key as an old master file record. Assume no "delete" transactions for the key.

1. Depending upon which record is input first, either the information from the transaction record or the information from the old master record will not be included in the resulting new master file.

2. The information from the transaction record will not be included in the resulting new master file.

3. The information from the old master record will not be included in the resulting new master file.

4. The information from the transaction record will be added to the new master file and then the information from the old master record will be added.

5. The information from the old master record will be added to the new master file and then the information from the transaction record will be added.

H. Assuming correct input data, what might be caused by changing line 540 to **TKEY .LE. OMKEY**?

1. An attempt may be made to input from a file after the end of the file has been detected.

2. An undefined variable may be referenced.

3. Unless both the transaction file and the old master file are empty, an end-of-file will not be detected for at least one of the two files.

4. 2 and 3 are correct

5. 1, 2, and 3 are correct

I. What will be caused by processing a "delete" transaction record with a key not matching any key found in the old master file or the rest of the transaction file? Assume that more than one, but not all, of the old master records have been input at the time the "bad" transaction record is input.

1. The information from the old master record input immediately before the "bad" transaction record is input may not be included in the resulting new master file even though it should be.

2. The information in the new master record currently being constructed may not be included in the resulting new master file even though it should be.

3. The information from the old master record input immediately after the "bad" transaction record is input may not be included in the resulting new master file even though it should be.

4. The answer will vary depending upon factors not stated.

5. none of the above
J. Assume that the old master file contains only four records and that the transaction file contains only two records, both "delete" transactions. Both of the files are correct except the records are in reverse order. How many records may the resulting new master file contain?

1. 2
2. 2 or 3
3. 3 or 4
4. 4
5. 2, 3, or 4

K. What might be caused by interchanging the keys of two randomly selected transaction records?

1. The resulting new master file may contain a record that it would not otherwise contain.
2. The resulting new master file may be missing a record that it would otherwise contain.
3. An update transaction ("replace", "debit", or "credit") may be applied to the wrong old master record.
4. 1 and 2 are correct
5. none of the above

L. What might be caused by interchanging the keys of two randomly selected old master records?

1. The resulting new master file may still be completely correct.
2. The resulting new master file will always contain at least one record that is out of sequence.
3. If "delete" transactions exist for both of the interchanged keys, then at least one of the transactions will always be properly processed.
4. 1 and 3 are correct
5. 2 and 3 are correct

M. Assume the old master file and the transaction file contain the indicated information. What sequence of values does OMKEY have while CURKEY has the value 32?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #)</td>
<td>(key)</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>(record #)</td>
<td>(key)</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>45</td>
</tr>
</tbody>
</table>

1. 20,32
2. 20,32,40
3. 32,40
4. 40,9999
5. none of the above
N. Assume the old master file and the transaction file contain the indicated information. What sequence of values does TKEY have while OMKEY has the value 30?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #) (key)</td>
<td>(record #) (key)</td>
</tr>
<tr>
<td>1 10</td>
<td>1 10</td>
</tr>
<tr>
<td>2 20</td>
<td>2 15</td>
</tr>
<tr>
<td>3 30</td>
<td>3 35</td>
</tr>
<tr>
<td>4 40</td>
<td>4 40</td>
</tr>
</tbody>
</table>

1. 15, 35, 40
2. 15, 35
3. 35, 40
4. 40
5. 35

D. Assume the old master file and the transaction file contain the indicated information. What sequence of values does NMKEY have while OMKEY has the value 30?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #) (key)</td>
<td>(record #) (code) (key)</td>
</tr>
<tr>
<td>1 10</td>
<td>1 3 10</td>
</tr>
<tr>
<td>2 20</td>
<td>2 1 25</td>
</tr>
<tr>
<td>3 30</td>
<td>3 3 30</td>
</tr>
<tr>
<td>4 40</td>
<td>4 3 30</td>
</tr>
</tbody>
</table>

1. 25, 30
2. 20, 25, 30
3. 10, 20, 25
4. 25, 30, 40
5. 30

TURN IN MATERIALS AS SOON AS YOU ARE FINISHED
APPENDIX H

Quiz for Hard Version of Program for Pilot Study 1

A. What might be caused by processing a transaction record with an invalid transaction code (TCODE)?
   1. The incorrect transaction record will have no effect on the resulting new master file.
   2. An infinite loop may be caused.
   3. The resulting new master file may contain a record it would not otherwise contain.
   4. The answer will vary depending upon factors not stated.
   5. The incorrect transaction record will have no effect on the resulting new master file only if it is the last record in the transaction file.

B. What is the total number of assignment statements executed if the old master file contains only one record and the transaction file is empty?
   1. 7
   2. 9
   3. 19
   4. 21
   5. none of the above

C. What is the maximum number of assignment statements that may be executed after OMK and TKEY both become equal to 9999?
   1. 3
   2. 7
   3. 11
   4. 1
   5. none of the above
D. Assuming correct input data, what might be caused by deleting line number 1460?

1. The resulting new master file may contain keys found in the transaction file and not in the old master file and may contain keys found in the old master file and not in the transaction file.
2. The resulting new master file will always be empty.
3. The resulting new master file will only contain keys that are also found in the old master file.
4. The resulting new master file will only contain keys that are also found in the transaction file.
5. None of the above

E. Assume the old master file and the transaction file contain the indicated information. How many times is the subprogram PADD called?

<table>
<thead>
<tr>
<th>OLD MASTER FILE (record #) (key)</th>
<th>TRANSACTION FILE (record #) (code) (key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 10</td>
<td>1 1 5</td>
</tr>
<tr>
<td>2 20</td>
<td>2 1 15</td>
</tr>
<tr>
<td>3 30</td>
<td>3 3 20</td>
</tr>
<tr>
<td>4 40</td>
<td>4 2 30</td>
</tr>
</tbody>
</table>

1. 8
2. 6
3. 2
4. 5
5. 4

F. What will be caused by processing a "credit" transaction record with a key not matching any key in the old master file or the rest of the transaction file?

1. An undefined variable may be referenced.
2. A new master record might have its balance incorrectly increased.
3. A new master record might have an undefined balance.
4. 1 and 3 are correct
5. 1, 2, and 3 are correct
G. What might be caused by processing an "add" transaction record having the same key as an old master file record. Assume no "delete" transactions for the key.

1. Depending upon which record is input first, either the information from the transaction record or the information from the old master record will not be included in the resulting new master file.

2. The information from the transaction record will not be included in the resulting new master file.

3. The information from the old master record will not be included in the resulting new master file.

4. The information from the transaction record will be added to the new master file and then the information from the old master record will be added.

5. The information from the old master record will be added to the new master file and then the information from the transaction record will be added.

H. Assuming correct input data, what might be caused by changing line 800 to TKEY, i.e. DMKEY?

1. An attempt may be made to input from a file after the end of the file has been detected.

2. An undefined variable may be referenced.

3. Unless both the transaction file and the old master file are empty, an end-of-file will not be detected for at least one of the two files.

4. 1, 2, and 3 are correct.

5. 2 and 3 are correct.

I. What will be caused by processing a "delete" transaction record with a key not matching any key found in the old master file or the rest of the transaction file? Assume that more than one, but not all, of the old master records have been input at the time the "bad" transaction record is input.

1. The information from the old master record input immediately before the "bad" transaction record is input may not be included in the resulting new master file even though it should be.

2. The information in the new master record currently being constructed may not be included in the resulting new master file even though it should be.

3. The information from the old master record input immediately after the "bad" transaction record is input may not be included in the resulting new master file even though it should be.

4. The answer will vary depending upon factors not stated.

5. None of the above.
J. Assume that the old master file contains only four records and that the transaction file contains only two records, both "delete" transactions. Both of the files are correct except the records are in reverse order. How many records may the resulting new master file contain?

1. 2
2. 2 or 3
3. 3 or 4
4. 2, 3, or 4
5. 4

K. What might be caused by interchanging the keys of two randomly selected transaction records?

1. The resulting new master file may contain a record that it would not otherwise contain.
2. The resulting new master file may be missing a record that it would otherwise contain.
3. An update transaction ("replace", "debit", or "credit") may be applied to the wrong old master record.
4. 1 and 2 are correct
5. none of the above

L. What might be caused by interchanging the keys of two randomly selected old master records?

1. The resulting new master file may still be completely correct.
2. If "delete" transactions exist for both of the interchanged keys, then at least one of the transactions will always be properly processed.
3. 1 and 2 are correct
4. The resulting new master file will always contain at least one record that is out of sequence.
5. 2 and 4 are correct
M. Assume the old master file and the transaction file contain the indicated information. What sequence of values does OMKEY have while CURKEY has the value 32?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #)</td>
<td>(record #)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

1. 20,32
2. 20,32,40
3. 40,9999
4. 32,40
5. none of the above

N. Assume the old master file and the transaction file contain the indicated information. What sequence of values does TKEY have while OMKEY has the value 30?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #)</td>
<td>(record #)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

1. 15,35,40
2. 15,35
3. 35,40
4. 40
5. 35

D. Assume the old master file and the transaction file contain the indicated information. What sequence of values does NMKEY have while DMKEY has the value 30?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #)</td>
<td>(record #)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

1. 25,30
2. 10,20,25
3. 20,25,30
4. 25,30,40
5. 30

TURN IN MATERIALS AS SOON AS YOU ARE FINISHED
APPENDIX I

Post-Experimental Questionnaire for All Pilot Studies and Experiments

POST-EXPERIMENTAL QUESTIONNAIRE

Name: _______________________________

A. On a scale from 1 to 5, how similar was the program used in this experiment to any other program you have ever written, studied, or worked with? (circle one number on the scale)

NOT SIMILAR 1 2 3 4 5 VERY SIMILAR

B. How well did you understand the program used in the experiment?

NOT WELL 1 2 3 4 5 VERY WELL

C. How easy was the program to understand?

NOT EASY 1 2 3 4 5 VERY EASY

D. How well written was the program?

NOT WELL WRITTEN 1 2 3 4 5 VERY WELL WRITTEN

E. What comments would you like to pass along concerning the program with which you have just worked? (Use space below)
# APPENDIX J

Means and Standard Deviations of Background Survey Responses and Post-Experimental Questionnaire Responses for Pilot Study 1

<table>
<thead>
<tr>
<th>Variable</th>
<th>Response</th>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Age</td>
<td></td>
<td>Easy</td>
<td>26.1</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>25.5</td>
<td>5.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>25.8</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. Sex</td>
<td></td>
<td>Easy</td>
<td>1.7</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>1.9</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>1.8</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. Class rank</td>
<td></td>
<td>Easy</td>
<td>4.8</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>4.6</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>4.7</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td></td>
<td>Easy</td>
<td>3.4</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>3.4</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>3.4</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. Number of programming courses taken</td>
<td></td>
<td>Easy</td>
<td>6.8</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>8.0</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>7.4</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td></td>
<td>Easy</td>
<td>3.4</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>3.6</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>3.5</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td></td>
<td>Easy</td>
<td>6.1</td>
<td>2.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>5.0</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>5.6</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td></td>
<td>Easy</td>
<td>.9</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>1.8</td>
<td>3.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>1.3</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I. Years of teaching</td>
<td></td>
<td>Easy</td>
<td>1.7</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>1.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Combined</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J. Years of programming</td>
<td></td>
<td>Easy</td>
<td>5.7</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>5.3</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>5.5</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td></td>
<td>Easy</td>
<td>4.1</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>4.1</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>4.1</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td></td>
<td>Easy</td>
<td>2.5</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>1.8</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>Variable</td>
<td>Program Type</td>
<td>Mean</td>
<td>Std Dev</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Easy</td>
<td>0.7</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>0.2</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>0.5</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Easy</td>
<td>2.1</td>
<td>2.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>1.4</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1.4</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>O. Years since writing in PL/1</td>
<td>Easy</td>
<td>0.7</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>0.2</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>0.6</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Easy</td>
<td>72.2</td>
<td>54.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>89.1</td>
<td>87.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>80.2</td>
<td>70.1</td>
<td></td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Easy</td>
<td>1800.0</td>
<td>1331.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>2293.8</td>
<td>2615.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>2032.4</td>
<td>1985.6</td>
<td></td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Easy</td>
<td>20.4</td>
<td>21.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>51.5</td>
<td>66.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>35.1</td>
<td>49.3</td>
<td></td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Easy</td>
<td>402.2</td>
<td>336.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>1284.4</td>
<td>2739.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>817.4</td>
<td>1083.3</td>
<td></td>
</tr>
<tr>
<td>T. Number of programs written in PL/1</td>
<td>Easy</td>
<td>21.8</td>
<td>19.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>13.0</td>
<td>18.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>17.6</td>
<td>19.1</td>
<td></td>
</tr>
<tr>
<td>U. Longest program in PL/1 (lines)</td>
<td>Easy</td>
<td>1111.1</td>
<td>990.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>562.5</td>
<td>724.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>852.9</td>
<td>894.5</td>
<td></td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Easy</td>
<td>5.8</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>6.1</td>
<td>3.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>5.9</td>
<td>2.9</td>
<td></td>
</tr>
<tr>
<td>W. WYLBUR experience 1 = low</td>
<td>Easy</td>
<td>2.4</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>5 = high</td>
<td>Hard</td>
<td>2.2</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>2.4</td>
<td>1.5</td>
<td></td>
</tr>
</tbody>
</table>
### Variable
**Post-Experimental Questionnaire**

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>2.9</td>
<td>1.5</td>
</tr>
<tr>
<td>Hard</td>
<td>2.8</td>
<td>1.3</td>
</tr>
<tr>
<td>Combined</td>
<td>2.8</td>
<td>1.3</td>
</tr>
</tbody>
</table>

1. Similarity of program with other programs

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>2.7</td>
<td>1.2</td>
</tr>
<tr>
<td>Hard</td>
<td>2.5</td>
<td>1.1</td>
</tr>
<tr>
<td>Combined</td>
<td>2.6</td>
<td>1.1</td>
</tr>
</tbody>
</table>

2. Degree of program understanding

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>2.1</td>
<td>1.2</td>
</tr>
<tr>
<td>Hard</td>
<td>2.0</td>
<td>1.1</td>
</tr>
<tr>
<td>Combined</td>
<td>2.1</td>
<td>1.1</td>
</tr>
</tbody>
</table>

3. Ease of program understanding

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>2.6</td>
<td>1.1</td>
</tr>
<tr>
<td>Hard</td>
<td>2.8</td>
<td>1.0</td>
</tr>
<tr>
<td>Combined</td>
<td>2.6</td>
<td>1.1</td>
</tr>
</tbody>
</table>

4. Degree to which program is well written

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>2.6</td>
<td>1.1</td>
</tr>
<tr>
<td>Hard</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Combined</td>
<td>2.6</td>
<td>1.1</td>
</tr>
</tbody>
</table>

* n = 9.

* Except where noted, n = 8 (one subject did not complete questionnaires).

* n = 6 since two subjects had no experience with Fortran. The corresponding subjects with no Fortran experience placed in the easy program group withdrew themselves from the experiment. The average score of the subjects with no Fortran experience was 2.5 while the average score of the subjects with some Fortran experience was 2.9.

* n = 4 since four subjects had no experience with PL/I.

* 1 = low, 5 = high.
### APPENDIX K

Correlations of Background Survey Responses and Post-Experimental Questionnaire Responses with Independent and Dependent Variables for Pilot Study 1

<table>
<thead>
<tr>
<th>Variable</th>
<th>Correlations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background Survey</strong></td>
<td><strong>Program Type</strong></td>
</tr>
<tr>
<td>Age</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Sex</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>1 = female</td>
</tr>
<tr>
<td></td>
<td>2 = male</td>
</tr>
<tr>
<td>Class rank</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>1 = freshman</td>
</tr>
<tr>
<td></td>
<td>5 = graduate</td>
</tr>
<tr>
<td>Overall GPA</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Number of programming courses taken</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Programming courses GPA</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Years of post high school education</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Years of professional programming</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Years of teaching programming</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Years of programming</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Years of programming in FORTRAN</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Years of programming in PL/1</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Variable</td>
<td>Background Survey</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>M. Years</td>
<td></td>
</tr>
<tr>
<td>writing a</td>
<td></td>
</tr>
<tr>
<td>program</td>
<td></td>
</tr>
<tr>
<td>N. Years</td>
<td>Easy</td>
</tr>
<tr>
<td>writing in</td>
<td>Hard</td>
</tr>
<tr>
<td>Fortran</td>
<td>Combined</td>
</tr>
<tr>
<td>O. Years</td>
<td>Easy</td>
</tr>
<tr>
<td>writing in</td>
<td>Hard</td>
</tr>
<tr>
<td>PL/I</td>
<td>Combined</td>
</tr>
<tr>
<td>P. Number</td>
<td>Easy</td>
</tr>
<tr>
<td>written</td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Q. Longest</td>
<td>Easy</td>
</tr>
<tr>
<td>program</td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>R. Number</td>
<td>Easy</td>
</tr>
<tr>
<td>written in</td>
<td>Hard</td>
</tr>
<tr>
<td>Fortran</td>
<td>Combined</td>
</tr>
<tr>
<td>S. Longest</td>
<td>Easy</td>
</tr>
<tr>
<td>program</td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>T. Number</td>
<td>Easy</td>
</tr>
<tr>
<td>written in</td>
<td>Hard</td>
</tr>
<tr>
<td>PL/I</td>
<td>Combined</td>
</tr>
<tr>
<td>U. Longest</td>
<td>Easy</td>
</tr>
<tr>
<td>program in</td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>V. Number</td>
<td>Easy</td>
</tr>
<tr>
<td>programming</td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>W. WYLBUR experience</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Variable</td>
<td>Correlations</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>Post-Experimental Questionnaire</td>
<td>Program Type</td>
</tr>
<tr>
<td>1. Similarity of program with other programs</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>2. Degree of program understanding</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>3. Ease of program understanding</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>4. Degree to which program is well written</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
</tbody>
</table>

*For purposes of correlation, easy program encoded as 1 and hard program encoded as 2.

\[ n = 9. \]

Except where noted, \( n = 8 \) (one subject did not complete questionnaires).

\[ n = 6 \] since two subjects had no experience with Fortran.

\[ n = 4 \] since four subjects had no experience with PL/I.

Coefficient could not be calculated since all subjects took the maximum time to complete the experiment.

\( 1 = \text{low}, 5 = \text{high}. \)

\[ p < .05. \]

\[ p < .01. \]
APPENDIX L

Consent Form for Volunteer Subjects

THE OHIO STATE UNIVERSITY  Protocol No. B280055

PRESENTATION TO SUBJECTS

You will be asked in this experiment to answer some questions about a program. We will note how long it takes you to answer these questions and the correctness of your answers. You will also be asked to fill out two questionnaires. One is concerned with your experience and aptitude in computer science and the other is concerned with your perception of the program with which you will work.

Your participation in the experiment is voluntary and you may withdraw at any time. You will sign-up for a two hour experimental session and will be paid $5.00 for each hour of participation in this session, up to a maximum of two hours.

The information you supply to us will be kept completely confidential.

From this experiment we hope to learn how to determine the complexity of computer software and thus possibly avoid unnecessary complexity.

CONSENT FOR PARTICIPATION IN RESEARCH

I consent to participation in research entitled "A Measure of the Psychological Complexity of Computer Software". Dr. Stuart Zweben or William E. Hall (Principal Investigators) has explained the purpose of the study, the procedures to be followed, and the expected duration of my participation. Possible benefits of the study have also been described.

I acknowledge that I have had the opportunity to obtain additional information regarding the study and that any questions I have raised have been answered to my full satisfaction. Further, I understand that I am free to withdraw consent at any time and to discontinue participation in the study without prejudice to me. I also understand that the information obtained from me will remain confidential.

Finally, I acknowledge that I have read and fully understand the consent form. I sign it freely and voluntarily. A copy has been given to me.

Date: ___________________________  Signed: ___________________________ (Participant)

Signed: ___________________________ (Principal Investigator)

Witness: ___________________________
APPENDIX M

Program Description for Pilot Study 2

PROGRAM DESCRIPTION FOR EXPERIMENT

Purpose: Update a master file
Input:  1. A master file (known as the old master file)
       2. A transaction file
Output: Another master file (known as the new master file)

Old master file description:

Each record consists of:

1. A bank account number (the key)
2. A bank balance
3. A name
4. An address

- Keys may range in value from 1 to 999
- No two records have the same key
- Records are in ascending key order

Transaction file description:

Each record consists of:

1. A transaction code
2. A bank account number (the key)
3. A transaction amount
4. A name
5. An address

- Keys may range in value from 1 to 999
- Records are in ascending key order. Records with the same key are in the order as they were created in real-time.

New master file description:

- Contains the same records as the old master file (and in the same format) except there may be some added, missing (deleted), or changed records as specified by transaction records.
- No two records have the same key
- Records are in ascending key order

Transactions code meanings:

<table>
<thead>
<tr>
<th>CODE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unless subsequent transactions specify otherwise, a record with the relevant information given in the transaction record is to be added to the new master file. Known as an &quot;add&quot; transaction.</td>
</tr>
<tr>
<td>2</td>
<td>Unless subsequent transactions specify otherwise, the information currently associated with the specified key is not to be included in the new master file. Known as a &quot;delete&quot; transaction.</td>
</tr>
</tbody>
</table>
3 If the name field of the transaction is not blank, then the name field of the new master file record currently associated with given key is to be changed to the indicated name. Similarly for the address field of the transaction record. Known as a "replace" transaction.

4 The balance field of the new master file record currently associated with the given key is to be decreased by the indicated amount. Known as a "debit" transaction.

5 The balance field of the new master file record currently associated with the given key is to be increased by the indicated amount. Known as a "credit" transaction.

Error processing:

There is no error detection - perfect input data is assumed.

Rational for program structure:

The program has been structured into several small routines to aid continued development. Routines that may be considered "too small" may be expected to grow as the problem specification grows in complexity.

Note: The program you will work with may have subroutines with multiple entry points (i.e., more than just the one standard entry point arrived at by CALLing the subroutine name). Each extra entry point is indicated via an ENTRY statement with an associated entry name and possible parameters. An ENTRY statement may be branched to by a CALL statement with the proper name. A RETURN instruction gets one back to the calling program. ENTRY statements do not in any way affect flow of control within a subroutine. Multiple entry points allow the programmer to group several related functions into one subroutine so that they may easily share data or code.

The program may also have variables that have been DECLARED STATIC instead of the default, AUTOMATIC. This means the variables will not be "destroyed" each time the procedure containing the variables is exited, but will remain for the next time the procedure is performed. For reasons of consistency some variables may have been DECLARED STATIC even though the program would work just as well if they had been DECLARED AUTOMATIC.
APPENDIX N

Listing of Easy Version of Program for Pilot Study 2

10. UPDATE:PROCEDURE OPTIONS(MAIN);
20. /*UPDATE FILE*/
40. DECLARE 1 TRANS REC   STATIC
60.  2 TRANS_CODE     FIXED(1,0),
70.  2 TRANS_INFO
80.  3 TRANS_KEY     FIXED(4,0),
90.  3 TRANS_AMOUNT  FIXED(7,2),
100. 3 TRANS_NAME    CHAR(35),
110. 3 TRANS_ADDRESS CHAR(35);
120. DECLARE 1 OLD MAST REC   STATIC
130.  2 ODD MAST KEY    FIXED(4,0),
140.  2 OLD MAST BALANCE FIXED(7,2),
150.  2 OLD MAST NAME  CHAR(35),
160.  2 OLD MAST ADDRESS CHAR(35);
170. DECLARE 1 NEW MAST REC   STATIC
180.  2 NEW MAST KEY    FIXED(4,0),
190.  2 NEW MAST BALANCE FIXED(7,2),
200.  2 NEW MAST NAME  CHAR(35),
210.  2 NEW MAST ADDRESS CHAR(35);
220. DECLARE CURRENT_KEY   STATIC FIXED(4,0);
230. DECLARE IN_USE   STATIC CHAR(5);
240. CALL IN_TRAN(TRANS REC);
260. CALL IN_MAST(OLD MAST REC);
270. CALL NEXT KEY(TRANS KEY,OLD MAST KEY,CURRENT KEY);
280. DO WHILE(CURRENT KEY ^= 9999);
290.    CALL PKEY(TRANS REC,OLD MAST REC,NEW MAST REC,IN_USE,
300.      CURRENT_KEY);
310. END;
320. IN_TRAN:PROCEDURE(TRANS REC);
330. /*READ A TRANSACTION RECORD*/
350. DECLARE 1 TRANS REC
370.  2 TRANS_CODE     FIXED(1,0),
380.  2 TRANS_INFO
390.  3 TRANS_KEY     FIXED(4,0),
400.  3 TRANS_AMOUNT  FIXED(7,2),
410.  3 TRANS_NAME    CHAR(*),
420.  3 TRANS_ADDRESS CHAR(*);
430. DECLARE TRANS FILE INPUT;
450. ON ENDFILE(TRANS) TRANS KEY = 9999;
460. GET FILE(TRANS) EDIT(TRANS REC)
470. (F(1),X(1),F(3),F(5,2),A(35),A(35));
490. RETURN;
500. END IN_TRAN;
IN_MAST:PROCEDURE(OLD_MAST_REC);

/*READ AN OLD MASTER RECORD*/

DECLARE 1 OLD_MAST_REC,

2 OLD_MAST_KEY   FIXED(4,0),

2 OLD_MAST_BALANCE   FIXED(7,2),

2 OLD_MAST_NAME      CHAR(*),

2 OLD_MAST_ADDRESS    CHAR(*);  

DECLARE OLDMAST_FILE INPUT;

ON ENDFILE(OLDMAST) OLD_MAST_KEY = 9999;
GET FILE(OLDMAST) EDIT(OLD_MAST_REC)

  (F(3),F(7,2),A(35),A(35));
RETURN;
END IN_MAST;

NEXT_KEY:PROCEDURE(TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY);

/*CHOOSE NEXT KEY*/

DECLARE (TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY) FIXED(4,0);

IF OLD_MAST_KEY <= TRANS_KEY
 THEN CURRENT_KEY = OLD_MAST_KEY;
ELSE CURRENT_KEY = TRANS_KEY;
RETURN;
END NEXT_KEY;
PROCEDURE \( \text{trans} \), \( \text{old mast rec} \), \( \text{new mast rec} \), \( \text{in use} \),
\( \text{current key} \);

/*PROCESS ONE KEY*/

DECLARE \( \text{trans rec} \),
\( \text{trans code} \) FIXED (1,0),
\( \text{trans info} \),
\( \text{trans key} \) FIXED (4,0),
\( \text{trans amount} \) FIXED (7,2),
\( \text{trans name} \) CHAR (*),
\( \text{trans address} \) CHAR (*);

DECLARE \( \text{old mast rec} \),
\( \text{old mast key} \) FIXED (4,0),
\( \text{old mast balance} \) FIXED (7,2),
\( \text{old mast name} \) CHAR (*),
\( \text{old mast address} \) CHAR (*);

DECLARE \( \text{new mast rec} \),
\( \text{new mast key} \) FIXED (4,0),
\( \text{new mast balance} \) FIXED (7,2),
\( \text{new mast name} \) CHAR (*),
\( \text{new mast address} \) CHAR (*);

DECLARE \( \text{in use} \),
\( \text{current key} \) FIXED (4,0);

CALL \( \text{init} \) (\( \text{new mast rec} \), \( \text{old mast rec} \), \( \text{in use} \), \( \text{current key} \));

DO WHILE (\( \text{trans key} \) = \( \text{current key} \));

CALL PTRANS (\( \text{trans rec} \), \( \text{new mast rec} \), \( \text{in use} \));

END;

CALL \( \text{final} \) (\( \text{new mast rec} \), \( \text{in use} \));

CALL \( \text{next key} \) (\( \text{trans key} \), \( \text{old mast key} \), \( \text{current key} \));

RETURN;

END \( \text{pkey} \);

INIT: PROCEDURE (\( \text{new mast rec} \), \( \text{old mast rec} \), \( \text{in use} \), \( \text{current key} \));

/*INITIAL STATUS*/

DECLARE \( \text{old mast rec} \),
\( \text{old mast key} \) FIXED (4,0),
\( \text{old mast balance} \) FIXED (7,2),
\( \text{old mast name} \) CHAR (*),
\( \text{old mast address} \) CHAR (*);

DECLARE \( \text{new mast rec} \),
\( \text{new mast key} \) FIXED (4,0),
\( \text{new mast balance} \) FIXED (7,2),
\( \text{new mast name} \) CHAR (*),
\( \text{new mast address} \) CHAR (*);

DECLARE \( \text{in use} \),
\( \text{current key} \) FIXED (4,0);

DECLARE \( \text{current key} \) FIXED (4,0);

IF \( \text{old mast key} \) = \( \text{current key} \)

THEN DO;

\( \text{new mast rec} \) = \( \text{old mast rec} \);
\( \text{in use} \) = "TRUE";

CALL \( \text{in mast} \) (\( \text{old mast rec} \));

END;

ELSE \( \text{in use} \) = "FALSE";

RETURN;

END INIT;
PTRAN5:PROCEDURE(TRANS_REC, NEW_MAST_REC, IN_USE);

/*PROCESS ONE TRANSACTION*/

DECLARE 1 TRANS_REC,
    2 TRANS_CODE    FIXED(1,0),
    2 TRANS_INFO,
    3 TRANS_KEY     FIXED(4,0),
    3 TRANS_AMOUNT  FIXED(7,2),
    3 TRANS_NAME    CHAR(*),
    3 TRANS_ADDRESS CHAR(*);

DECLARE 1 NEW_MAST_REC,
    2 NEW_MAST_KEY FIXED(4,0),
    2 NEW_MAST_BALANCE FIXED(7,2),
    2 NEW_MAST_NAME CHAR(*),
    2 NEW_MAST_ADDRESS CHAR(*);

DECLARE IN_USE
    CHAR(*);

CALL MODIFY(TRANS_INFO, NEW_MAST_REC, IN_USE);
CALL IN_TRNS(TRANS_REC);
RETURN;
END PTRAN;

FINAL:PROCEDURE(NEW_MAST_REC, IN_USE);

/*FINAL STATUS*/

DECLARE 1 NEW_MAST_REC,
    2 NEW_MAST_KEY FIXED(4,0),
    2 NEW_MAST_BALANCE FIXED(7,2),
    2 NEW_MAST_NAME CHAR(*),
    2 NEW_MAST_ADDRESS CHAR(*);

DECLARE IN_USE
    CHAR(*);

DECLARE NEW_MAST_FILE OUTPUT;

IF IN_USE = 'TRUE'
THEN PUT FILE(NEW_MAST) EDIT(NEW_MAST_REC)
(COL(1), X(1), F(3), X(1), F(7,2), X(1), A(35), A(35));
RETURN;
END FINAL;
2150.  MODIFY:PROCEDURE (TRANS_INFO,NEW_MAST_REC,IN_USE);
2160.  /*MODIFY NEW MASTER*/
2170.  DECLARE 1 TRANS_INFO,
2180.       2 TRANS_KEY                  FIXED(4,0),
2190.       2 TRANS_AMOUNT               FIXED(7,2),
2200.       2 TRANS_NAME                 CHAR(*),
2210.       2 TRANS_ADDRESS             CHAR(*);
2220.  DECLARE 1 NEW_MAST_REC,
2230.       2 NEW_MAST_KEY               FIXED(4,0),
2240.       2 NEW_MAST_BALANCE           FIXED(7,2),
2250.       2 NEW_MAST_NAME              CHAR(*),
2260.       2 NEW_MAST_ADDRESS           CHAR(*);
2270.  DECLARE IN_USE                   CHAR(*);
2280.  IF TRANS_CODE = 1
2290.      THEN CALL PADD (TRANS_INFO,NEW_MAST_REC,IN_USE);
2300.  ELSE IF TRANS_CODE = 2
2310.      THEN CALL PDELETE (IN_USE);
2320.  ELSE IF TRANS_CODE = 3
2330.      THEN CALL PREPLACE (TRANS_NAME,TRANS_ADDRESS,
2340.          NEW_MAST_NAME,NEW_MAST_ADDRESS);
2350.  ELSE IF TRANS_CODE = 4
2360.      THEN CALL PDEBIT (TRANS_AMOUNT,NEW_MAST_BALANCE);
2370.  ELSE IF TRANS_CODE = 5
2380.      THEN CALL PCREDIT (TRANS_AMOUNT,NEW_MAST_BALANCE);
2390.  RETURN;
2400.  END MODIFY;
2410.  PADD:PROCEDURE (TRANS_INFO,NEW_MAST_REC,IN_USE);
2420.  /*PROCESS DIFFERENT MODIFICATIONS*/
2430.  DECLARE 1 TRANS_INFO,
2440.       2 KEY                      FIXED(4,0),
2450.       2 AMOUNT                   FIXED(7,2),
2460.       2 NAME                     CHAR(*),
2470.       2 ADDRESS                  CHAR(*);
2480.  DECLARE 1 NEW_MAST_REC,
2490.       2 KEY                      FIXED(4,0),
2500.       2 BALANCE                 FIXED(7,2),
2510.       2 NAME                     CHAR(*),
2520.       2 ADDRESS                  CHAR(*);
2530.  DECLARE (IN_USE,TRANS_NAME,TRANS_ADDRESS,
2540.          NEW_MAST_NAME,NEW_MAST_ADDRESS) CHAR(*);
2550.  DECLARE (TRANS_AMOUNT,NEW_MAST_BALANCE) FIXED(7,2);
2560.  /*ADDITION*/
2570.  NEW_MAST_REC = TRANS_INFO;
2580.  IN_USE = 'TRUE';
2590.  RETURN;
2600.  PDELETE:ENTRY (IN_USE);
2610.  /*DELETION*/
2620.  IN_USE = 'FALSE';
2630.  RETURN;
PREPLACE:ENTRY(TRANS_NAME, TRANS_ADDRESS, NEW_MAST_NAME, NEW_MAST_ADDRESS);

REPLACEMENT/

IF TRANS_NAME ' ' THEN NEW_MAST_NAME = TRANS_NAME;
IF TRANSADDRESS ' ' THEN NEW_MAST_ADDRESS = TRANS_ADDRESS;
RETURN;

PDEBIT:ENTRY(TRANS_AMOUNT, NEW_MAST_BALANCE);

DEBIT/

NEW_MAST_BALANCE = NEW_MAST_BALANCE - TRANS_AMOUNT;
RETURN;

PCREDIT:ENTRY(TRANS_AMOUNT, NEW_MAST_BALANCE);

CREDIT/

NEW_MAST_BALANCE = NEW_MAST_BALANCE + TRANS_AMOUNT;
RETURN;
END PADD;
END UPDATE;
APPENDIX O

Listing of Hard Version of Program for Pilot Study 2

10. UPDATE:PROCEDURE OPTIONS(MAIN);
20. /UPDATE FILE*/
30. DECLARE INFO_REC
40. 2 KEY
50. FIXED(4,0),
60. 2 AMOUNT
70. FIXED(7,2),
80. 2 NAME
90. CHAR(35),
100. 2 ADDRESS
110. CHAR(35);
120. DECLARE END_FLAG
130. STATIC CHAR(5);
140. CALL BEGIN_IN;
150. CALL BEGIN_OUT;
160. CALL GET_INFO(INFO_REC,END_FLAG);
170. DO WHILE END_FLAG = "FALSE";
180. CALL PREC(INFO_REC,END_FLAG);
190. END;
200. CALL END_OUT;
210. 
220. 
230. 
240. PREC:PROCEDURE(INFO_REC,END_FLAG);
250. /PROCESS ONE RECORD*/
260. DECLARE INFO_REC,
270. 2 KEY
280. FIXED(4,0),
290. 2 AMOUNT
300. FIXED(7,2),
310. 2 NAME
320. CHAR(*),
330. 2 ADDRESS
340. CHAR(*);
350. DECLARE END_FLAG
360. STATIC CHAR(5);
370. DECLARE NEW
380. STATIC CHAR(5);
390. DECLARE CODE
400. STATIC FIXED(1,0);
410. CALL NEW_GRP(NEW);
420. IF NEW = 'TRUE'
430. THEN CALL NEW KEY;
440. CALL PTYPE(CODE);
450. IF CODE = 1
460. THEN CALL PADD(INFO_REC);
470. ELSE IF CODE = 2
480. THEN CALL PDELETE;
490. ELSE IF CODE = 3
500. THEN CALL PREPLACE(NAME,ADDRESS);
510. ELSE IF CODE = 4
520. THEN CALL PDEBIT(AMOUNT);
530. ELSE IF CODE = 5
540. THEN CALL PCREDIT(AMOUNT);
550. CALL GET_INFO(INFO_REC,END_FLAG);
560. RETURN;
570. END PREC;
BEGIN_IN:PROCEDURE;

/*INPUT MODULE*/

/*INITIALIZE INPUT*/

DECLARE 1 TRANS REC  STATIC
   2 TRANS_CODE   FIXED(1,0),
   2 TRANS_INFO   FIXED(4,0),
   3 TRANS_KEY    FIXED(4,0),
   3 TRANS_AMOUNT FIXED(7,2),
   3 TRANS_NAME   CHAR(35),
   3 TRANS_ADDR   CHAR(35);

DECLARE 1 OLD MAST REC  STATIC
   2 OCD MAST_KEY  FIXED(4,0),
   2 OLD_MAST_BAL  FIXED(7,2),
   2 OLD_MAST_NAME CHAR(35),
   2 OLD_MAST_ADDR CHAR(35);

DECLARE 1 INFO REC,  STATIC
   2 KEY        FIXED(4,0),
   2 AMOUNT     FIXED(7,2),
   2 NAME       CHAR(*),
   2 ADDRESS    CHAR(*);

DECLARE CURRENT_KEY   STATIC  FIXED(4,0);

DECLARE CODE    FIXED(1,0);

DECLARE TYPE    STATIC  FIXED(1,0);

DECLARE NEW FLAG  STATIC  CHAR(5);

DECLARE (NEW,END_FLAG)  CHAR(*);

DECLARE TRANS FILE INPUT;

DECLARE OLD MAST FILE INPUT;

ON ENDFILE (TRANS) TRANS KEY = 9999;

ON ENDFILE (OLD MAST) OLD_MAST KEY = 9999;

GET FILE (TRANS) EDIT (TRANS REC)
   (F(1),F(3),F(5,2),A(35),A(35));

GET FILE (OLD MAST) EDIT (OLD MAST REC)
   (F(3),F(7,2),A(35),A(35));

CURRENT KEY = 0;

NEW FLAG = 'FALSE';

RETURN;

1000.
1010.
1020. GET_INFO:ENTRY(INFO_REC,END_FLAG);
1030. ON ENDFILE(TRANS) TRANS_KEY = 9999;
1040. ON ENDFILE(OLDMAST) OLD_MAST_KEY = 9999;
1050. /*GET ONE RECORD*/
1060. IF (OLD_MAST_KEY = 9999) & (TRANS_KEY = 9999)
1070. THEN DD:
1080. END_FLAG = 'TRUE';
1090. RETURN;
1100. END;
1110. END_FLAG = 'FALSE';
1120. IF OLD_MAST_KEY <= TRANS_KEY
1130. THEN DD:
1140. INFO_REC = OLD_MAST_REC;
1150. TYPE = 1;
1160. GET_FILE(OLDMAST) EDIT(OLDMAST_REC)
1170. (F(3),F(7,2),A(35),A(35));
1180. END;
1190. ELSE DD:
1200. INFO_REC = TRANS_INFO;
1210. TYPE = TRANS_CODE;
1220. GET_FILE(TRANS) EDIT(TRANS_REC)
1230. (F(1),X(1),F(3),F(5,2),A(35),A(35));
1240. END;
1250. IF KEY = CURRENT_KEY
1260. THEN NEW_FLAG = 'FALSE';
1270. ELSE DD:
1280. NEW_FLAG = 'TRUE';
1290. CURRENT_KEY = KEY;
1300. END;
1310. RETURN;
1320. /*GET ENTRY*/
1330. PTYPE:ENTRY(CODE);
1340. / *GET PROCESSING TYPE*/
1350. CODE = TYPE;
1360. RETURN;
1370. NEW_GRP:ENTRY(NEW);
1380. /*GET GROUP STATUS*/
1390. NEW = NEW_FLAG;
1400. RETURN;
1410. END BEGIN_IN;
BEGIN_OUT:PROCEDURE;
/*OUTPUT MODULE*/
/*INITIALIZE OUTPUT*/
DECLARE 1 INFO_REC,
   2 KEY, FIXED(4,0),
   2 AMOUNT, FIXED(7,2),
   2 NAME, CHAR(*),
   2 ADDRESS, CHAR(*);
DECLARE 1 NEW_MAST_REC, STATIC,
   2 NEW_MAST_KEY, FIXED(4,0),
   2 NEW_MAST_BALANCE, FIXED(7,2),
   2 NEW_MAST_NAME, CHAR(35),
   2 NEW_MAST_ADDRESS, CHAR(35);
DECLARE (TRANS_NAME,TRANS_ADDRESS) CHAR(*);
DECLARE TRANS_AMOUNT, FIXED(7,2);
DECLARE BUFFER, STATIC CHAR(5);
DECLARE NEWMAST FILE OUTPUT;
ENDPROCEDURE;

PADD:ENTRY(INFO_REC);
/*PROCESS ADDITION*/
NEW_MAST_REC = INFO_REC;
BUFFER = 'FULL';
RETURN;

PDELETE:ENTRY;
/*PROCESS DELETION*/
BUFFER = 'EMPTY';
RETURN;

PREPLACE:ENTRY(TRANS_NAME,TRANS_ADDRESS);
/*PROCESS REPLACE*/
IF TRANS_NAME ^= '' THEN NEW_MAST_NAME = TRANS_NAME;
IF TRANS_ADDRESS ^= '' THEN NEW_MAST_ADDRESS = TRANS_ADDRESS;
RETURN;

PDEBIT:ENTRY(TRANS_AMOUNT);
/*PROCESS DEBIT*/
NEW_MAST_BALANCE = NEW_MAST_BALANCE - TRANS_AMOUNT;
RETURN;

PCREDIT:ENTRY(TRANS_AMOUNT);
/*CREDIT*/
NEW_MAST_BALANCE = NEW_MAST_BALANCE + TRANS_AMOUNT;
RETURN;
NEW_KEY:ENTRY;

/*PROCESS A NEW KEY*/

IF BUFFER = 'FULL'
  THEN DO:
    PUT FILE(NEWMAST) EDIT(NEW_MAST_REC)
      (COL(1),X(1),F(3),X(1),F(7,2),X(L),A(35),A(35));
    BUFFER = 'EMPTY';
  END;
RETURN;

END_OUT:ENTRY;

/*FINALIZE OUTPUT PROCESSING*/

IF BUFFER = 'FULL'
  THEN PUT FILE(NEWMAST) EDIT(NEW_MAST_REC)
    (COL(1),X(1),F(3),X(1),F(7,2),X(L),A(35),A(35));
RETURN;

END_BEGIN_OUT;
END_UPDATE;
APPENDIX P

Quiz for Easy Version of Program for Pilot Study 2

A. What might be caused by processing a transaction record with an invalid transaction code (TRANS_CODE)?

1. The incorrect transaction record will have no effect on the resulting new master file.
2. An infinite loop may be caused.
3. The resulting new master file may contain a record it would not otherwise contain.
4. The answer will vary depending upon factors not stated.
5. The incorrect transaction record will have no effect on the resulting new master file only if it is the last record in the transaction file.

B. What is the total number of assignment statements executed if the old master file contains only one record and the transaction file is empty?

1. 4
2. 15
3. 13
4. 6
5. none of the above

C. What is the maximum number of assignment statements that may be executed after OLD_MAST_KEY and TRANS_KEY both become equal to 9999?

1. 3
2. 7
3. 1
4. 5
5. none of the above
D. Assuming correct input data, what might be caused by deleting line number 2710?

1. The resulting new master file may contain keys found in the transaction file and not in the old master file and may contain keys found in the old master file and not in the transaction file.
2. The resulting new master file will only contain keys that are also found in the old master file.
3. The resulting new master file will always be empty.
4. The resulting new master file will only contain keys that are also found in the transaction file.
5. none of the above

E. Assume the old master file and the transaction file contain the indicated information. How many times is the subprogram PADD called?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #) (key)</td>
<td>(record #) (code) (key)</td>
</tr>
<tr>
<td>1 10</td>
<td>1 1 5</td>
</tr>
<tr>
<td>2 20</td>
<td>2 1 15</td>
</tr>
<tr>
<td>3 30</td>
<td>3 3 20</td>
</tr>
<tr>
<td>4 40</td>
<td>4 2 30</td>
</tr>
</tbody>
</table>

1. 8
2. 2
3. 6
4. 5
5. 4

F. What will be caused by processing a "credit" transaction record with a key not matching any key in the old master file or the rest of the transaction file?

1. An undefined variable may be referenced.
2. A new master record might have its balance incorrectly increased.
3. A new master record might have an undefined balance.
4. 1 and 3 are correct
5. 1, 2, and 3 are correct
6. What might be caused by processing an "add" transaction record having the same key as an old master file record. Assume no "delete" transactions for the key.

1. Depending upon which record is input first, either the information from the transaction record or the information from the old master record will not be included in the resulting new master file.

2. The information from the transaction record will not be included in the resulting new master file.

3. The information from the old master record will not be included in the resulting new master file.

4. The information from the transaction record will be added to the new master file and then the information from the old master record will be added.

5. The information from the old master record will be added to the new master file and then the information from the transaction record will be added.

H. Assuming correct input data, what might be caused by changing line 870 to TRANS_KEY <= OLD_MAST_KEY?

1. A attempt may be made to input from a file after the end of the file has been detected.

2. An undefined variable may be referenced.

3. Unless both the transaction file and the old master file are empty, an end-of-file will not be detected for at least one of the two files.

4. 2 and 3 are correct.

5. 1, 2, and 3 are correct.

I. What will be caused by processing a "delete" transaction record with a key not matching any key found in the old master file or the rest of the transaction file? Assume that more than one, but not all, of the old master records have been input at the time the "bad" transaction record is input.

1. The information from the old master record input immediately before the "bad" transaction record is input may not be included in the resulting new master file even though it should be.

2. The information in the new master record currently being constructed may not be included in the resulting new master file even though it should be.

3. The information from the old master record input immediately after the "bad" transaction record is input may not be included in the resulting new master file even though it should be.

4. The answer will vary depending upon factors not stated.

5. None of the above.
J. Assume that the old master file contains only four records and that the transaction file contains only two records, both "delete" transactions. Both of the files are correct except the records are in reverse order. How many records may the resulting new master file contain?

1. 2
2. 2 or 3
3. 3 or 4
4. 4
5. 2, 3, or 4

K. What might be caused by interchanging the keys of two randomly selected transaction records?

1. The resulting new master file may contain a record that it would not otherwise contain.
2. The resulting new master file may be missing a record that it would otherwise contain.
3. An update transaction ("replace", "debit", or "credit") may be applied to the wrong old master record.
4. 1 and 2 are correct
5. none of the above

L. What might be caused by interchanging the keys of two randomly selected old master records?

1. The resulting new master file may still be completely correct.
2. The resulting new master file will always contain at least one record that is out of sequence.
3. If "delete" transactions exist for both of the interchanged keys, then at least one of the transactions will always be properly processed.
4. 1 and 3 are correct
5. 2 and 3 are correct

M. Assume the old master file and the transaction file contain the indicated information. What sequence of values does OLD_MAST_KEY have while CURRENT_KEY has the value 32?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #) (key)</td>
<td>(record #) (key)</td>
</tr>
<tr>
<td>1  10</td>
<td>1  10</td>
</tr>
<tr>
<td>2  20</td>
<td>2  22</td>
</tr>
<tr>
<td>4  32</td>
<td>3  32</td>
</tr>
<tr>
<td>4  40</td>
<td>4  45</td>
</tr>
</tbody>
</table>

1. 20,32
2. 20,32,40
3. 32,40
4. 40,9999
5. none of the above
N. Assume the old master file and the transaction file contain the indicated information. What sequence of values does TRAN$\_KEY$ have while OLD$\_MAST$\_KEY has the value 30?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #)</td>
<td>(record #)</td>
</tr>
<tr>
<td>(key)</td>
<td>(key)</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
</tbody>
</table>

1. 15, 35, 40
2. 15, 35
3. 35, 40
4. 40
5. 35

0. Assume the old master file and the transaction file contain the indicated information. What sequence of values does NEW$\_MAST$\_KEY have while OLD$\_MAST$\_KEY has the value 30?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #)</td>
<td>(record #)</td>
</tr>
<tr>
<td>(key)</td>
<td>(code) (key)</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
</tbody>
</table>

1. 25, 30
2. 20, 25, 30
3. 10, 20, 25
4. 25, 30, 40
5. 30
APPENDIX Q
Quiz for Hard Version of Program for Pilot Study 2

A. What might be caused by processing a transaction record with an invalid transaction code (TRANS_CODE)?

1. The incorrect transaction record will have no effect on the resulting new master file.
2. An infinite loop may be caused.
3. The resulting new master file may contain a record it would otherwise contain.
4. The answer will vary depending upon factors not stated.
5. The incorrect transaction record will have no effect on the resulting new master file only if it is the last record in the transaction file.

B. What is the total number of assignment statements executed if the old master file contains only one record and the transaction file is empty?

1. 4
2. 6
3. 13
4. 15
5. none of the above

C. What is the maximum number of assignment statements that may be executed after OLD_MAST_KEY and TRANS_KEY both become equal to 9999?

1. 3
2. 5
3. 8
4. 1
5. none of the above
D. Assuming correct input data, what might be caused by deleting line number 1870?

1. The resulting new master file may contain keys found in the transaction file and not in the old master file and may contain keys found in the old master file and not in the transaction file.

2. The resulting new master file will always be empty.

3. The resulting new master file will only contain keys that are also found in the old master file.

4. The resulting new master file will only contain keys that are also found in the transaction file.

5. none of the above

E. Assume the old master file and the transaction file contain the indicated information. How many times is the subprogram PADD called?

<table>
<thead>
<tr>
<th>OLD MASTER FILE</th>
<th>TRANSACTION FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(record #)</td>
<td>(key)</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>(record #)</td>
<td>(code)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

1. 8
2. 6
3. 2
4. 5
5. 4

F. What will be caused by processing a "credit" transaction record with a key not matching any key in the old master file or the rest of the transaction file?

1. An undefined variable may be referenced.

2. A new master record might have its balance incorrectly increased.

3. A new master record might have an undefined balance.

4. 1 and 3 are correct

5. 1, 2, and 3 are correct
G. What might be caused by processing an "add" transaction record having the same key as an old master file record. Assume no "delete" transactions for the key.

1. Depending upon which record is input first, either the information from the transaction record or the information from the old master record will not be included in the resulting new master file.

2. The information from the transaction record will not be included in the resulting new master file.

3. The information from the old master record will not be included in the resulting new master file.

4. The information from the transaction record will be added to the new master file and then the information from the old master record will be added.

5. The information from the old master record will be added to the new master file and then the information from the transaction record will be added.

H. Assuming correct input data, what might be caused by changing line 1150 to `TRANS_KEY <= OLD_MAST_KEY`?

1. An attempt may be made to input from a file after the end of the file has been detected.

2. An undefined variable may be referenced.

3. Unless both the transaction file and the old master file are empty, an end-of-file will not be detected for at least one of the two files.

4. 1, 2, and 3 are correct

5. 2 and 3 are correct

I. What will be caused by processing a "delete" transaction record with a key not matching any key found in the old master file or the rest of the transaction file? Assume that more than one, but not all, of the old master records have been input at the time the "bad" transaction record is input.

1. The information from the old master record input immediately before the "bad" transaction record is input may not be included in the resulting new master file even though it should be.

2. The information in the new master record currently being constructed may not be included in the resulting new master file even though it should be.

3. The information from the old master record input immediately after the "bad" transaction record is input may not be included in the resulting new master file even though it should be.

4. The answer will vary depending upon factors not stated.

5. none of the above
J. Assume that the old master file contains only four records and that the transaction file contains only two records, both "delete" transactions. Both of the files are correct except the records are in reverse order. How many records may the resulting new master file contain?
1. 2
2. 2 or 3
3. 3 or 4
4. 2, 3, or 4
5. 4

K. What might be caused by interchanging the keys of two randomly selected transaction records?
1. The resulting new master file may contain a record that it would not otherwise contain.
2. The resulting new master file may be missing a record that it would otherwise contain.
3. An update transaction ("replace", "debit", or "credit") may be applied to the wrong old master record.
4. 1 and 2 are correct
5. none of the above

L. What might be caused by interchanging the keys of two randomly selected old master records?
1. The resulting new master file may still be completely correct.
2. If "delete" transactions exist for both of the interchanged keys, then at least one of the transactions will always be properly processed.
3. 1 and 2 are correct
4. The resulting new master file will always contain at least one record that is out of sequence.
5. 2 and 4 are correct

M. Assume the old master file and the transaction file contain the indicated information. What sequence of values does OLD_MAST_KEY have while CURRENT_KEY has the value 32?

OLD MASTER FILE
(record #) (key)
1 10
2 20
3 32
4 40

TRANSACTION FILE
(record #) (key)
1 10
2 22
3 32
4 45

1. 20,32
2. 20,32,40
3. 40,9999
4. 32,40
5. none of the above
N. Assume the old master file and the transaction file contain the indicated information. What sequence of values does TRANS_KEY have while OLD_MAST_KEY has the value 30?

<table>
<thead>
<tr>
<th>OLD MASTER FILE (record #) (key)</th>
<th>TRANSACTION FILE (record #) (key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 10</td>
<td>1 10</td>
</tr>
<tr>
<td>2 20</td>
<td>2 15</td>
</tr>
<tr>
<td>3 30</td>
<td>3 35</td>
</tr>
<tr>
<td>4 40</td>
<td>4 40</td>
</tr>
</tbody>
</table>

1. 15, 35, 40
2. 15, 35
3. 35, 40
4. 40
5. 35

D. Assume the old master file and the transaction file contain the indicated information. What sequence of values does NEW_MAST_KEY have while OLD_MAST_KEY has the value 30?

<table>
<thead>
<tr>
<th>OLD MASTER FILE (record #) (key)</th>
<th>TRANSACTION FILE (record #) (code) (key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 30</td>
<td>1 3 10</td>
</tr>
<tr>
<td>2 20</td>
<td>2 1 25</td>
</tr>
<tr>
<td>3 30</td>
<td>3 3 30</td>
</tr>
<tr>
<td>4 40</td>
<td>4 3 30</td>
</tr>
</tbody>
</table>

1. 25, 30
2. 10, 20, 25
3. 20, 25, 30
4. 25, 30, 40
5. 30

TURN IN MATERIALS AS SOON AS YOU ARE FINISHED
## APPENDIX R

Means and Standard Deviations of Background Survey Responses and Post-Experimental Questionnaire Responses for Pilot Study 2

<table>
<thead>
<tr>
<th>Variable</th>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background Survey</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. Age</td>
<td>Easy</td>
<td>23.2</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>21.4</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>22.3</td>
<td>3.2</td>
</tr>
<tr>
<td>B. Sex</td>
<td>Easy</td>
<td>1.8</td>
<td>.4</td>
</tr>
<tr>
<td>1 = female</td>
<td>Hard</td>
<td>1.7</td>
<td>.5</td>
</tr>
<tr>
<td>2 = male</td>
<td>Combined</td>
<td>1.7</td>
<td>.5</td>
</tr>
<tr>
<td>C. Class rank</td>
<td>Easy</td>
<td>3.9</td>
<td>.6</td>
</tr>
<tr>
<td>1 = freshman</td>
<td>Hard</td>
<td>3.7</td>
<td>.5</td>
</tr>
<tr>
<td>5 = graduate</td>
<td>Combined</td>
<td>3.8</td>
<td>.5</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td>Easy</td>
<td>2.8</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.0</td>
<td>.2</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>2.9</td>
<td>.4</td>
</tr>
<tr>
<td>E. Number of programming courses taken</td>
<td>Easy</td>
<td>7.2</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>6.3</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>6.8</td>
<td>1.8</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td>Easy</td>
<td>3.2</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.3</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.2</td>
<td>.4</td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td>Easy</td>
<td>4.4</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.6</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.0</td>
<td>1.1</td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td>Easy</td>
<td>.4</td>
<td>.9</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.2</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.3</td>
<td>.7</td>
</tr>
<tr>
<td>I. Years of teaching programming</td>
<td>Easy</td>
<td>.0</td>
<td>.0</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.0</td>
<td>.0</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.0</td>
<td>.0</td>
</tr>
<tr>
<td>J. Years of programming</td>
<td>Easy</td>
<td>3.1</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.2</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.2</td>
<td>1.7</td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td>Easy</td>
<td>1.1</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.5</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.8</td>
<td>1.5</td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td>Easy</td>
<td>2.3</td>
<td>.8</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>2.0</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>2.2</td>
<td>.6</td>
</tr>
<tr>
<td>Variable</td>
<td>Program Type</td>
<td>Mean</td>
<td>Std Dev</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>Background Survey</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Easy</td>
<td>.2</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.0</td>
<td>.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.1</td>
<td>.3</td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Easy</td>
<td>0.8</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>1.7</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1.1</td>
<td>1.7</td>
</tr>
<tr>
<td>O. Years since writing in PL/I</td>
<td>Easy</td>
<td>.2</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.1</td>
<td>.3</td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Easy</td>
<td>39.7</td>
<td>13.0</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>56.2</td>
<td>55.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>47.9</td>
<td>39.8</td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Easy</td>
<td>1405.6</td>
<td>1160.1</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>2133.3</td>
<td>3008.7</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1769.4</td>
<td>2243.6</td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Easy</td>
<td>7.3</td>
<td>7.0</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>2.8</td>
<td>6.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>5.1</td>
<td>7.0</td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Easy</td>
<td>461.1</td>
<td>975.6</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>48.9</td>
<td>86.7</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>235.0</td>
<td>704.6</td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Easy</td>
<td>26.3</td>
<td>10.7</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>27.8</td>
<td>12.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>27.1</td>
<td>11.4</td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Easy</td>
<td>1077.8</td>
<td>927.4</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>1311.1</td>
<td>690.8</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1194.4</td>
<td>890.3</td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Easy</td>
<td>4.4</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.8</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.1</td>
<td>1.3</td>
</tr>
<tr>
<td>W. WYLBUR experience 1 = low</td>
<td>Easy</td>
<td>3.8</td>
<td>.7</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.3</td>
<td>.7</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.6</td>
<td>.7</td>
</tr>
<tr>
<td>Variable</td>
<td>Program Type</td>
<td>Mean</td>
<td>Std Dev</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>--------------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>Post-Experimental Questionnaire</td>
<td>Easy</td>
<td>3.7</td>
<td>.7</td>
</tr>
<tr>
<td>1. Similarity of program with other programs</td>
<td>Hard</td>
<td>3.8</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.7</td>
<td>.9</td>
</tr>
<tr>
<td>2. Degree of program understanding</td>
<td>Easy</td>
<td>3.6</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.6</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.6</td>
<td>.5</td>
</tr>
<tr>
<td>3. Ease of program understanding</td>
<td>Easy</td>
<td>3.0</td>
<td>.9</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>2.8</td>
<td>.7</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>2.9</td>
<td>.6</td>
</tr>
<tr>
<td>4. Degree to which program is well written</td>
<td>Easy</td>
<td>3.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>2.9</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>2.9</td>
<td>1.1</td>
</tr>
</tbody>
</table>

*Except where noted, \( n = 9 \).*

*\( n = 6 \) since three subjects had no experience with Fortran.*

*\( n = 3 \) since six subjects had no experience with Fortran.*

*1 = low, 5 = high.*
### APPENDIX S

**Correlations of Background Survey Responses and Post-Experimental Questionnaire Responses with Independent and Dependent Variables for Pilot Study 2**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Correlations</th>
<th>Program Type</th>
<th>Score</th>
<th>Time</th>
<th>Prog*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Survey</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. Age</td>
<td></td>
<td>Easy</td>
<td>-.37</td>
<td>.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard*</td>
<td>.68</td>
<td>.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.08</td>
<td>.05</td>
<td>-.28</td>
</tr>
<tr>
<td>B. Sex</td>
<td></td>
<td>Easy</td>
<td>-.39</td>
<td>.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.59</td>
<td>.42</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.27</td>
<td>.26</td>
<td>-.12</td>
</tr>
<tr>
<td>C. Class rank</td>
<td></td>
<td>Easy</td>
<td>-.01</td>
<td>.36</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>-.34</td>
<td>-.38</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>-.07</td>
<td>-.14</td>
<td>-.21</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td></td>
<td>Easy</td>
<td>.38</td>
<td>-.42</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.16</td>
<td>.59</td>
<td>.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.13</td>
<td>.08</td>
<td></td>
</tr>
<tr>
<td>E. Number of programming</td>
<td></td>
<td>Easy</td>
<td>-.17</td>
<td>-.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.22</td>
<td>.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.13</td>
<td>-.28</td>
<td>-.26</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td></td>
<td>Easy</td>
<td>.47</td>
<td>-.40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.58</td>
<td>.34</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.33</td>
<td>-.04</td>
<td>.09</td>
</tr>
<tr>
<td>G. Years of post high</td>
<td></td>
<td>Easy</td>
<td>-.30</td>
<td>.04</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.41</td>
<td>-.16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.29</td>
<td>-.25</td>
<td>-.39</td>
</tr>
<tr>
<td>H. Years of professional</td>
<td></td>
<td>Easy</td>
<td>-.01</td>
<td>-.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>-.07</td>
<td>-.17</td>
<td>-.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.07</td>
<td>-.26</td>
<td></td>
</tr>
<tr>
<td>I. Years of teaching</td>
<td></td>
<td>Easy</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>J. Years of programming</td>
<td></td>
<td>Easy</td>
<td>.67</td>
<td>-.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.74</td>
<td>.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.63</td>
<td>.11</td>
<td>.02</td>
</tr>
<tr>
<td>K. Years of programming in</td>
<td></td>
<td>Easy</td>
<td>.49</td>
<td>-.24</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.05</td>
<td>.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.30</td>
<td>-.07</td>
<td>-.24</td>
</tr>
<tr>
<td>L. Years of programming in</td>
<td></td>
<td>Easy</td>
<td>.60</td>
<td>-.13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.59</td>
<td>.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.57</td>
<td>-.10</td>
<td>-.28</td>
</tr>
<tr>
<td>Variable</td>
<td>Background Survey</td>
<td>Program Type</td>
<td>Score</td>
<td>Time</td>
<td>Prog</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>--------------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Easy</td>
<td>-.18</td>
<td>-.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.19</td>
<td>.25</td>
<td>.06</td>
<td>-.24</td>
<td>-.25</td>
</tr>
<tr>
<td>Combined</td>
<td>.06</td>
<td>-.24</td>
<td>.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Easy</td>
<td>-.37</td>
<td>-.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.97</td>
<td>.64</td>
<td>.44</td>
<td>.16</td>
<td>.26</td>
</tr>
<tr>
<td>Combined</td>
<td>.44</td>
<td>.16</td>
<td>.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O. Years since writing in PL/I</td>
<td>Easy</td>
<td>-.13</td>
<td>-.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.12</td>
<td>.28</td>
<td>.10</td>
<td>-.22</td>
<td>-.28</td>
</tr>
<tr>
<td>Combined</td>
<td>.10</td>
<td>-.22</td>
<td>.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Easy</td>
<td>.09</td>
<td>.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.75</td>
<td>.28</td>
<td>.46</td>
<td>.30</td>
<td>.21</td>
</tr>
<tr>
<td>Combined</td>
<td>.46</td>
<td>.30</td>
<td>.21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Easy</td>
<td>-.08</td>
<td>.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.01</td>
<td>-.55</td>
<td>-.08</td>
<td>-.18</td>
<td>.17</td>
</tr>
<tr>
<td>Combined</td>
<td>-.08</td>
<td>-.18</td>
<td>.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Easy</td>
<td>.25</td>
<td>-.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.12</td>
<td>.34</td>
<td>.29</td>
<td>-.10</td>
<td>-.33</td>
</tr>
<tr>
<td>Combined</td>
<td>.29</td>
<td>-.10</td>
<td>-.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Easy</td>
<td>-.33</td>
<td>.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>-.06</td>
<td>.28</td>
<td>-.01</td>
<td>-.00</td>
<td>-.30</td>
</tr>
<tr>
<td>Combined</td>
<td>-.01</td>
<td>-.00</td>
<td>-.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Easy</td>
<td>-.10</td>
<td>.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.63</td>
<td>-.15</td>
<td>.31</td>
<td>-.00</td>
<td>.07</td>
</tr>
<tr>
<td>Combined</td>
<td>.31</td>
<td>-.00</td>
<td>.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Easy</td>
<td>-.09</td>
<td>.67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.30</td>
<td>-.33</td>
<td>.07</td>
<td>.13</td>
<td>.13</td>
</tr>
<tr>
<td>Combined</td>
<td>.07</td>
<td>.13</td>
<td>.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Easy</td>
<td>.37</td>
<td>-.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.41</td>
<td>-.24</td>
<td>.45</td>
<td>-.05</td>
<td>-.26</td>
</tr>
<tr>
<td>Combined</td>
<td>.45</td>
<td>-.05</td>
<td>-.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W. WYLBUR experience</td>
<td>Easy</td>
<td>-.14</td>
<td>.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = low</td>
<td>.14</td>
<td>.33</td>
<td>-.01</td>
<td>-.06</td>
<td>-.32</td>
</tr>
<tr>
<td>Hard</td>
<td>-.22</td>
<td>-.02</td>
<td>.14</td>
<td>.33</td>
<td>-.01</td>
</tr>
</tbody>
</table>
### Variable Correlations

<table>
<thead>
<tr>
<th>Post-Experimental* Questionnaire</th>
<th>Program Type</th>
<th>Score</th>
<th>Time</th>
<th>Prog</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Similarity of program with other programs</td>
<td>Easy</td>
<td>.07</td>
<td>.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.11</td>
<td>.63*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.06</td>
<td>.47*</td>
<td>.06</td>
</tr>
<tr>
<td>2. Degree of program understanding</td>
<td>Easy</td>
<td>-.07</td>
<td>-.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.19</td>
<td>.61*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.08</td>
<td>.19</td>
<td>.00</td>
</tr>
<tr>
<td>3. Ease of program understanding</td>
<td>Easy</td>
<td>-.27</td>
<td>.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>-.03</td>
<td>.16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.04</td>
<td>.16</td>
<td>-.15</td>
</tr>
<tr>
<td>4. Degree to which program is well written</td>
<td>Easy</td>
<td>.00</td>
<td>.43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.19</td>
<td>-.27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.14</td>
<td>-.04</td>
<td>-.05</td>
</tr>
</tbody>
</table>

*1 = low, 5 = high.

**For purposes of correlation, easy program encoded as 1 and hard program encoded as 2.

*b = 9.

**Coefficient cannot be calculated because survey response is constant for all cases (no one had taught programming).

**b = 6 since three subjects had no experience with Fortran.

**b = 3 since six subjects had no experience with Fortran.

**1 = low, 5 = high.

**p < .05.

**p < .01.
APPENDIX I

Instruction Sheet for Cloze Experiments

INSTRUCTIONS

On the next pages you will find the description and listing of a program. The program listing is missing some pieces. Each missing piece is a single symbol such as an operator, operand, word, number, etc. The missing pieces have been replaced by short lines. Each line represents one missing symbol. All of the lines are of equal length and therefore do not indicate the size of the missing pieces. Your task is to fill in the missing pieces on the program listing. You have approximately an hour and forty-five minutes. Please work as quickly and accurately as possible. Raise your hand whenever you have a question.

Note that not everyone is working with the same materials and some may finish before others. Turn in this booklet as soon as you have finished filling in the missing pieces.
APPENDIX U
Program Description for Cloze Experiments

PROGRAM DESCRIPTION FOR EXPERIMENT

Purpose: Update a master file
Input: 1. A master file (known as the old master file)
       2. A transaction file
Output: Another master file (known as the new master file)

Old master file description:
Each record consists of:

<table>
<thead>
<tr>
<th>Column</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>F(7)</td>
</tr>
<tr>
<td>2.</td>
<td>F(7,2)</td>
</tr>
<tr>
<td>3.</td>
<td>A(35)</td>
</tr>
<tr>
<td>4.</td>
<td>A(35)</td>
</tr>
</tbody>
</table>

- Keys may range in value from 1 to 999
- No two records have the same key
- Records are in ascending key order

Transaction file description:
Each record consists of:

<table>
<thead>
<tr>
<th>Column</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>F(1)</td>
</tr>
<tr>
<td>2.</td>
<td>F(3)</td>
</tr>
<tr>
<td>3.</td>
<td>F(5,2)</td>
</tr>
<tr>
<td>4.</td>
<td>A(35)</td>
</tr>
<tr>
<td>5.</td>
<td>A(35)</td>
</tr>
</tbody>
</table>

- Keys may range in value from 1 to 999
- Records are in ascending key order. Records with the same key are in the order as they were created in real-time.

New master file description:
- Contains the same records as the old master file (and in the same format) except there may be some added, missing (deleted), or changed records as specified by transaction records.
- No two records have the same key
- Records are in ascending key order
Transactions code meanings:

<table>
<thead>
<tr>
<th>CODE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unless subsequent transactions specify otherwise, a record with the relevant information given in the transaction record is to be added to the new master file. Known as an &quot;add&quot; transaction.</td>
</tr>
<tr>
<td>2</td>
<td>Unless subsequent transactions specify otherwise, the information currently associated with the specified key is not to be included in the new master file. Known as a &quot;delete&quot; transaction.</td>
</tr>
<tr>
<td>3</td>
<td>If the name field of the transaction is not blank, then the name field of the new master file record currently associated with given key is to be changed to the indicated name. Similarly for the address field of the transaction record. Known as a &quot;replace&quot; transaction.</td>
</tr>
<tr>
<td>4</td>
<td>The balance field of the new master file record currently associated with the given key is to be decreased by the indicated amount. Known as a &quot;debit&quot; transaction.</td>
</tr>
<tr>
<td>5</td>
<td>The balance field of the new master file record currently associated with the given key is to be increased by the indicated amount. Known as a &quot;credit&quot; transaction.</td>
</tr>
</tbody>
</table>

Error processing:

There is no error detection - perfect input data is assumed.

Rational for program structure:

The program has been structured into several small routines to aid continued development. Routines that may be considered "too small" may be expected to grow as the problem specification grows in complexity.

Note:

The program you will work with may have subroutines with multiple entry points (i.e., more than just the one standard entry point arrived at by CALLing the subroutine name). Each extra entry point is indicated via an ENTRY statement with an associated entry name and possible parameters. An ENTRY statement may be branched to by a CALL statement with the proper name. A RETURN instruction gets one back to the calling program. ENTRY statements do not in any way affect flow of control within a subroutine. Multiple entry points allow the programmer to group several related functions into one subroutine so that they may easily share data or code.

The program may also have variables that have been DECLARED STATIC instead of the default, AUTOMATIC. This means the variables will not be "destroyed" each time the procedure containing the variables is exited, but will remain for the next time the procedure is performed. For reasons of consistency some variables may have been DECLARED STATIC even though the program would work just as well if they had been DECLARED AUTOMATIC.
APPENDIX V

Easy Program Version Cloze Test for Experiment 1

1. "="PROCEDURE OPTIONS(MAIN);
/UPDATE FILE*/

2. DECLARE 1 TRANS_REC
   2 TRANS_CODE FIXED(10),
   2 TRANS_INFO,
   3 TRANS_KEY FIXED(4),
   3 TRANS_AMOUNT FIXED(7),
   3 TRANS_NAME CHAR(35),
   3 TRANS_ADDRESS CHAR(35);

3. DECLARE 1 OLDMAST_REC STATIC
   2 FIXED(4),
   2 FIXED(7),
   2 CHAR(35),
   2 OLD'MAST'ADDRESS CHAR(35);

4. DECLARE 1 NEWMAST_REC STATIC
   2 NEWMAST_KEY FIXED(4),
   2 NEWMAST_BALANCE FIXED(7),
   2 NEWMAST_NAME CHAR(35),
   2 NEW'MAST'ADDRESS CHAR(35);

5. DECLARE CURRENT_KEY STATIC
   4,0;

6. DECLARE IN_USE STATIC
   4,0;

7. CALL IN_TRAN(TRANS_REC);

8. CALL IN' mast( )

9. DO WHILE(TRANS_KEY,OLDMAST_KEY,CURRENT_KEY)

10. CALL PRETTYTRANS_REC -------,NEWMAST_REC,
    IN_USE,CURRENT_KEY)

11. END;

12. IN_TRAN:PROCEDURE( );
/READ A TRANSACTION RECORD*/

13. DECLARE 1 TRANS_REC
   2 TRANS_CODE FIXED(10),
   2 TRANS_INFO,
   TRANS_KEY FIXED(4),
   TRANS_AMOUNT FIXED(7),
   TRANS_NAME CHAR(*),
   TRANS_ADDRESS CHAR(*);

14. DECLARE TRANS_FILE;

15. ON ENDFILE(TRANS) TRANS_KEY =
   GET FILE(TRANS) EDIT(TRANS_REC);

16. RETURN;
END IN_TRAN;

195
IN_MAST: PROCEDURE (__________________________);

/*READ AN OLD MASTER RECORD*/
DECLARE 1 OLD_MAST_REC,
  2 OCD_MAST_KEY (4,0),
  2 OLD_MAST_BALANCE (7,2),
  2 OLD_MAST_NAME (*),
  2 OLD_MAST_ADDRESS-CHAR(__________________________);
DECLARE OLD_MAST_FILE_INPUT;
ON_ENDFILE(__________________________) OLD_MAST_KEY = 9999;
GET_FILE(__________________________) EDIT(OLD_MAST_REC)
  (__________________________) A(35),
RETURN;
END_IN_MAST;

NEXT_KEY: PROCEDURE (TRANS_KEY,__________________________,CURRENT_KEY);
/*CHOOSE NEXT KEY*/
DECLARE (TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY)
  (__________________________) (4,0);
IF OLD_MAST_KEY <=
  THEN CURRENT_KEY = OLD_MAST_KEY;
ELSE CURRENT_KEY = TRANS_KEY;
RETURN;
END_NEXT_KEY;

PKEY: PROCEDURE (TRANS_REC,__________________________,NEW_MAST_REC,
  IN_USE,CURRENT_KEY);
/*PROCESS ONE KEY*/
DECLARE 1
  2 TRANS_CODE-FIXED1,__________________________;
  2 TRANS_INFO,
  3 TRANS_KEY (4,0),
  3 TRANS_AMOUNT (7,2),
  3 TRANS_NAME (*),
  3 TRANS_ADDRESS-CHAR(__________________________);
DECLARE 1 OLD_MAST_REC,
  2 OCD_MAST_KEY (4,0),
  2 OLD_MAST_BALANCE (7,2),
  2 OLD_MAST_NAME (*),
  2 OLD_MAST_ADDRESS-CHAR(__________________________);
DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY (4,0),
  2 NEW_MAST_BALANCE (7,2),
  2 NEW_MAST_NAME (*),
  2 NEW_MAST_ADDRESS-CHAR(__________________________);
DECLARE CURRENT_KEY FIXED(11);0;
DECLARE ________________ CHAR(*);
CALL INIT(NEW_MAST_REC,__________________________,IN_USE,CURRENT_KEY);
DO WHILE(TRANS_KEY =)
  CALL_TRNS(TRANS_REC,NEW_MAST_REC,IN_USE);
END;
CALL
CALL NEXT_KEY(TRANS_KEY,__________________________,CURRENT_KEY);
RETURN;
END PKEY;
65. INIT: PROCEDURE (NEW_MAST_REC, ---------------, IN_USE, CURRENT_KEY);

       /* INITIAL STATUS */

DECLARE 1 OLD_MAST_REC,
       --------------------- OLD_MAST_KEY FIXED (4, 0),
66. --------------------- OLD_MAST_BALANCE FIXED (7, 2),
67. 2 --------------------- OLD_MAST_NAME CHAR (*),
68. 2
69. DECLARE 1 NEW_MAST_REC;
70. 2 --------------------- NEW_MAST_KEY FIXED (4, 0),
71. 2 --------------------- NEW_MAST_BALANCE FIXED (7, 2),
72. 2
73. 2
74. DECLARE CURRENT_KEY FIXED (-------------------, 0);
75. DECLARE IN_USE CHAR (*);
76. IF *CURRENT_KEY
77. THEN DO;------------------ = CURRENT_KEY
78. NEW_MAST_REC =
79. IN_USE = 'TRUE';-------------;
80. CALL IN_MAST(-------------------);
81. ELSE IN_USE = 'FALSE';
82. RETURN;
83. END;
84. END INIT;

85. ------------------: PROCEDURE (TRANS_REC, NEW_MAST_REC, IN_USE);

       /* PROCESS ONE TRANSACTION */

86. DECLARE 1
87. 2 TRANS_CODE FIXED (71;---------------------),
88. 2 TRANS_INFO,
89. 3 TRANS_KEY
90. 3 TRANS_AMOUNT FIXED (7, 0),
91. 3 TRANS_NAME
92. 3 TRANS_ADDRESS CHAR (*);
93. DECLARE 1 NEW_MAST_REC,
94. 2 NEW_MAST_KEY FIXED (4, 0),
95. 2 NEW_MAST_BALANCE FIXED (7, 2),
96. 2 NEW_MAST_NAME CHAR (*),
97. 2 NEW_MAST_ADDRESS CHAR (*);
98. DECLARE IN_USE CHAR (*);
99. CALL MODIFY(-------------------, NEW_MAST_REC, IN_USE);
100. CALL IN_TRNSTRANS_RECT;
101. RETURN;
102. END;------------------;
FINAL: PROCEDURE(NEW_MAST_REC, IN_USE);

/*FINAL STATUS*/

91. DECLARE 1
92. 2 NEW_MAST_KEY FIXED(4),
93. 2 NEW_MAST_BALANCE FIXED(7),
94. 2 NEW_MAST_NAME CHAR(*),
95. 2 NEW_MAST_ADDRESS CHAR(*);
96. DECLARE IN_USE,
97. DECLARE NEW_MAST_REC INPUT
98. DECLARE NEWMASTER OUTPUT (`);
99. IF POT_FIX(NEW_MAST_REC) = 'TRUE'
100. THEN CALL POT_FIX(NEW_MAST_REC);
101. RETURN;
102. END FINAL;

MODIFY: PROCEDURE(NEW_MAST_REC, IN_USE);

/*MODIFY NEW MASTER*/

DECLARE 1 TRANS_REC,
103. TRANS_CODE FIXED(1, 0),
104. TRANS_KEY FIXED(7, 0),
105. TRANS_AMOUNT FIXED(7, 2),
106. TRANS_NAME CHAR(*),
107. TRANS_ADDRESS CHAR(*);
108. DECLARE 1 NEW_MAST_REC,
109. 2 NEW_MAST_KEY FIXED(4),
110. 2 NEW_MAST_BALANCE FIXED(7),
111. 2 NEW_MAST_NAME CHAR(*),
112. 2 NEW_MAST_ADDRESS CHAR(*);
113. IF TRANS_CODE =
114. THEN CALL PADD(TRANS_INFO, NEW_MAST_REC, IN_USE);
115. ELSE IF = 2
116. THEN CALL PDELETE(NEW_MAST_REC);
117. ELSE IF = 3
118. THEN CALL PREPLACE(TRANS_NAME, NEW_MAST_NAME, NEW_MAST_ADDRESS);
119. ELSE IF TRANS_CODE =
120. THEN CALL PBRI_TD(TRANS_AMOUNT, NEW_MAST_BALANCE);
121. ELSE IF TRANS_CODE =
122. THEN CALL PB_PCE_TD(TRANS_AMOUNT, NEW_MAST_BALANCE);
123. RETURN;
124. END.
PADD: PROCEDURE (TRANS_INFO, NEW_MAST_REC, IN_USE);

/*PROCESS DIFFERENT MODIFICATIONS*/

118. DECLARE TRANS_INFO,
119.  2 KEY          FIXED (---------------, 0),
120.  2 AMOUNT      FIXED (---------------, 2),
121.  2 NAME        CHAR (---------------T),
122.  2 ADDRESS     CHAR (---------------T),
123. DECLARE NEW_MAST_REC,
124.  2 KEY          FIXED (---------------, 0),
125.  2 BALANCE     FIXED (---------------, 2),
126.  2 NAME        CHAR (---------------T),
127.  2 ADDRESS     CHAR (---------------T),
128. DECLARE (TRANS_NAME, TRANS_ADDRESS,
129.  NEW_MAST_NAME, NEW_MAST_ADDRESS) (*);
130. DECLARE (TRANS_AMOUNT, NEW_MAST_BALANCE) (*);

/*ADDICTION*/

131. NEW_MAST_REC = TRANS_INFO;
132. RETURN;
133. PDELETE: ENTRY (IN_USE);

/*DELETION*/

134. RETURN; = 'FALSE';
135. PREPLACE: ENTRY (TRANS_NAME, NEW_MAST_NAME,
136. NEW_MAST_ADDRESS);

/*REPLACEMENT*/

137. IF TRANS_NAME ^= TRANS_NAME THEN NEW_MAST_NAME ^= TRANS_NAME;
138. IF TRANS_ADDRESS ^= TRANS_ADDRESS THEN NEW_MAST_ADDRESS ^= TRANS_ADDRESS;
139. RETURN;
140. 
141. DEBIT: ENTRY (TRANS_AMOUNT, NEW_MAST_BALANCE);

/*DEBIT*/

142. NEW_MAST_BALANCE = NEW_MAST_BALANCE - TRANS_AMOUNT;
143. RETURN;
144. PCREDIT: ENTRY (TRANS_AMOUNT, NEW_MAST_BALANCE);

/*CREDIT*/

145. NEW_MAST_BALANCE = NEW_MAST_BALANCE + TRANS_AMOUNT;
146. RETURN;
147. END PADD;
148. END UPDATE;

PLEASE TURN IN THIS BOOKLET AS SOON AS YOU ARE FINISHED
APPENDIX W

Hard Program Version Close Test for Experiment I

1. --------------- : PROCEDURE OPTIONS(MAIN);
   /*UPDATE FILE*/
2. DECLARE 1 INFO_REC
3.     2 KEY             FIXED(4,0);
4.     2 AMOUNT         FIXED(7,2);
5.     2 NAME           CHAR(35);
6.     DECLARE END_FLAG ADDRESS CHAR(35);
    CALL BEGIN_IN;
    CALL BEGIN_OUT;
7.    CALL (INFO_REC,END_FLAG);
8.    DO WHILE(END_FLAG = (INFO_REC,END_FLAG)) ;
    CALL PREC (INFO_REC,END_FLAG) ;
    END;
    CALL END_OUT;

9. --------------- : PROCEDURE (INFO_REC,END_FLAG);
   /*PROCESS ONE RECORD*/
10. DECLARE 1 INFO_REC,
    --------------- KEY  FIXED(4,0),
    --------------- AMOUNT FIXED(7,2),
    NAME       CHAR(*);
11. DECLARE END_FLAG CHAR(*) ;
12. DECLARE NEW STATIC CHAR(*) ;
13. DECLARE STATIC FIXED(1,0);
14. CALL NEW (NEW);
15. IF NEW = 'TRUE' THEN CALL P SpaceX;
16. IF CODE = THEN CALL P Sleep;
17. ELSE IF CODE = ' ' THEN CALL P Sleep;
18. ELSE IF CODE = 3 THEN CALL P Sleep;
19. ELSE IF CODE = ' ' THEN CALL P Sleep;
20. ELSE IF CODE = THEN CALL P Sleep;
21. ELSE IF CODE = THEN CALL P Sleep;
22. ELSE IF CODE = THEN CALL P Sleep;
23. CALL GET_INFO (INFO_REC,END_FLAG)
   RETURN;
   END PREC;
BEGIN_PROCEDURE;
/* INPUT MODULE */
/* INITIALIZE INPUT */

24. DECLARE TRANS REC
25. 2 TRANS_CODE FIXED(11,11),
    2 TRANS_INFO,
26. 3 TRANS_KEY (4,0),
27. 3 TRANS_AMOUNT (7,2),
28. 3 TRANS_NAME (35),
29. 3 TRANS_ADDRESS CHAR,

30. DECLARE OLD_MAST REC STATIC
31. 2 ------------------ FIXED(4,0),
32. 2 ------------------ FIXED(7,2),
33. 2 ------------------ CHAR(35),
34. 2 OLD_MAST_ADDRESS (35);

35. DECLARE INFO REC,
36. 2 ------------------ FIXED(4,0),
37. 2 ------------------ FIXED(7,2),
38. 2 ------------------ CHAR(*),
39. 2 ADDRESS (*)

40. DECLARE CURRENT_KEY STATIC FIXED(10);
41. DECLARE TYPE STATIC FIXED(11,11);
42. DECLARE NEW_FLAG STATIC CHAR(5),
43. DECLARE END_FLAG CHAR(*);

44. ON TRANS.TRAN_KEY = 9999;
45. ON OLD_MAST.OLD_MAST_KEY = 9999;
46. GET TRANS EDIT(TRANS REC)
47. ------------------- (5,2), A(35),
48. ------------------- (35));
49. GET FILE(OLD_MAST EDIT(TRANS REC)
50. ------------------- (5,2), A(35),
51. ------------------- (35));
52. REW_FLAG = 0;
RETURN;

GET_INFO: ENTRY(INFO_REC, END_FLAG);

ON ENDFILE(________) TRANS_KEY = 9999;

ON ENDFILE(________) OLD_MAST_KEY = 9999;

/* GET ONE RECORD */

IF (OLD_MAST_KEY = 9999) & (TRANS_KEY = 9999) THEN DO;
  END_FLAG = 'TRUE';
  RETURN;
END;

IF OLD_MAST_KEY = TRANS_KEY THEN DO;
  TYPE = OLD_MAST_REC;
  GET_FILE(OLD_MASTREC) EDIT(OLD_MAST_REC)
  (________) (3), F(7,2), (________) (35), A(35));
  END;
ELSE DO;
  INFO_REC TYPE = TRANS_CODE;
  GET (TRANS) EDIT(TRANS_REC)
  (________) (3), F(7,2), A(35), (________) (35));
  END;
  IF KEY = CURRENT KEY THEN
  END_FLAG = 'FALSE';
ELSE DD;
  NEW_FLAG = CURRENT_KEY = KEY;
  END;
  RETURN;

PTYPE: ENTRY(________);

/* GET PROCESSING TYPE */

CODE = TYPE;
RETURN;

NEW_GRP: ENTRY(________);

/* GET GROUP STATUS */

NEW = NEW_FLAG;
RETURN;
END BEGIN_IN;
PROCEDURE:

/*OUTPUT MODULE*/
/*INITIALIZE OUTPUT*/

DECLARE 1 INFO_REC,
  2 KEY (4,0);
DECLARE 1 NEW_MAST_REC
  2 NAME (7,2),
  2 ADDRESS CHAR(35) ;
DECLARE 1 NEW_MAST_REC STATIC 7
  2 FIXED(4,0),
  2 FIXED(7,2),
  2 CHAR(35),
  2 NEW_MAST_ADDRESS CHAR(35);

ENTRY (INFO.REC);

NEW_MAST_REC = INFO_REC;
RETURN;

PDELETE:ENTRY;

PREPLACE:ENTRY(TRANS_NAME,TRANS_ADDRESS);

IF
  THEN NEW_MAST_ADDRESS = TRANS_ADDRESS;
RETURN;

PDEBIT:ENTRY(TRANS_AMOUNT);

NEW_MAST_BALANCE = NEW_MAST_BALANCE - TRANS_AMOUNT;
RETURN;

PCREDIT:ENTRY(TRANS_AMOUNT);

NEW_MAST_BALANCE = NEW_MAST_BALANCE + TRANS_AMOUNT;
RETURN;
NEW_KEY:ENTRY;
/*PROCESS A NEW KEY*/

IF BUFFER = 'FULL'
THEN DO:

   PUT ------------------------ EDIT(NEW_MAST)

   (COL(------------------------),F(3),
   F(7,------------------------),A(35),A(35));

   END;----------------------------- = "EMPTY";

   RETURN;

END_OUT:ENTRY;

/*FINALIZE OUTPUT PROCESSING*/

IF BUFFER 'FULL'
THEN PUT FILENEW_MAST]EDIT("FULL"

   (COL(1),F(3),------------------------77,2),
   A(35),------------------------735));

RETURN;
END BEGIN OUT;
END UPDATE;

PLEASE TURN IN THIS BOOKLET AS SOON AS YOU ARE FINISHED.
APPENDIX X

Means and Standard Deviations of Background Survey Responses and Post-Experimental Questionnaire Responses for Experiment 1

<table>
<thead>
<tr>
<th>Variable</th>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Survey</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. Age</td>
<td>Easy</td>
<td>24.4</td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>21.8</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>23.1</td>
<td>3.9</td>
</tr>
<tr>
<td>B. Sex</td>
<td>Easy</td>
<td>1.5</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Male</td>
<td>1.7</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1.6</td>
<td>.5</td>
</tr>
<tr>
<td>C. Class rank</td>
<td>Easy</td>
<td>3.5</td>
<td>.8</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.6</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.6</td>
<td>.9</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td>Easy</td>
<td>3.2</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>2.8</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.0</td>
<td>.6</td>
</tr>
<tr>
<td>E. Number of programming courses</td>
<td>Easy</td>
<td>5.8</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>6.3</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>6.0</td>
<td>1.9</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td>Easy</td>
<td>3.3</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.1</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.2</td>
<td>.5</td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td>Easy</td>
<td>4.4</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>4.1</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.3</td>
<td>2.0</td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td>Easy</td>
<td>.1</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.4</td>
<td>.7</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.3</td>
<td>.5</td>
</tr>
<tr>
<td>I. Years of teaching programming</td>
<td>Easy</td>
<td>.0</td>
<td>.0</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.3</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.2</td>
<td>.4</td>
</tr>
<tr>
<td>J. Years of programming</td>
<td>Easy</td>
<td>3.5</td>
<td>3.4</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.0</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.2</td>
<td>2.5</td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td>Easy</td>
<td>1.7</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.2</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.9</td>
<td>2.8</td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td>Easy</td>
<td>2.8</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>2.3</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>2.6</td>
<td>2.3</td>
</tr>
<tr>
<td>Variable</td>
<td>Background Survey</td>
<td>Program Type</td>
<td>Response</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Easy</td>
<td>.0</td>
<td>.0</td>
</tr>
<tr>
<td>Hard</td>
<td>.0</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>.0</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Easy</td>
<td>1.2</td>
<td>1.7</td>
</tr>
<tr>
<td>Hard</td>
<td>1.5</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>1.3</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>O. Years since writing in PL/I</td>
<td>Easy</td>
<td>.1</td>
<td>.3</td>
</tr>
<tr>
<td>Hard</td>
<td>.1</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>.1</td>
<td>.2</td>
<td></td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Easy</td>
<td>47.6</td>
<td>28.5</td>
</tr>
<tr>
<td>Hard</td>
<td>51.4</td>
<td>28.5</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>49.5</td>
<td>27.9</td>
<td></td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Easy</td>
<td>1504.5</td>
<td>1360.2</td>
</tr>
<tr>
<td>Hard</td>
<td>1886.4</td>
<td>1499.3</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>1695.5</td>
<td>1410.6</td>
<td></td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Easy</td>
<td>13.2</td>
<td>29.1</td>
</tr>
<tr>
<td>Hard</td>
<td>2.7</td>
<td>3.8</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>8.0</td>
<td>20.9</td>
<td></td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Easy</td>
<td>210.9</td>
<td>433.8</td>
</tr>
<tr>
<td>Hard</td>
<td>325.0</td>
<td>894.9</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>268.0</td>
<td>688.8</td>
<td></td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Easy</td>
<td>30.1</td>
<td>23.6</td>
</tr>
<tr>
<td>Hard</td>
<td>25.4</td>
<td>13.5</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>27.7</td>
<td>19.0</td>
<td></td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Easy</td>
<td>1437.3</td>
<td>1383.6</td>
</tr>
<tr>
<td>Hard</td>
<td>1318.2</td>
<td>1477.0</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>1377.7</td>
<td>1397.9</td>
<td></td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Easy</td>
<td>4.1</td>
<td>1.7</td>
</tr>
<tr>
<td>Hard</td>
<td>4.5</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>4.3</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>W. WYLBUR experience</td>
<td>Easy</td>
<td>3.4</td>
<td>1.1</td>
</tr>
<tr>
<td>Hard</td>
<td>3.4</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>3.4</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Response</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Post-Experimental Questionnaire</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Program Type</strong></td>
<td><strong>Mean</strong></td>
<td><strong>Std Dev</strong></td>
<td></td>
</tr>
<tr>
<td>1. Similarity of program with other programs</td>
<td>Easy</td>
<td>4.0</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>4.0</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.0</td>
<td>1.2</td>
</tr>
<tr>
<td>2. Degree of program understanding</td>
<td>Easy</td>
<td>4.5</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.8</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>3. Ease of program understanding</td>
<td>Easy</td>
<td>4.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.0</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.5</td>
<td>1.1</td>
</tr>
<tr>
<td>4. Degree to which program is well written</td>
<td>Easy</td>
<td>3.7</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.2</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.5</td>
<td>1.9</td>
</tr>
</tbody>
</table>

*Except where noted, n = 11.

$n = 7$ since four subjects had no experience with Fortran.

$n = 5$ since six subjects had no experience with Fortran.

$1 = $ low, $5 = $ high.
APPENDIX V

Correlations of Background Survey Responses and Post-Experimental Questionnaire Responses with Program Version and Synonymic Score for Experiment 1

<table>
<thead>
<tr>
<th>Variable</th>
<th>Background Survey</th>
<th>Program Type</th>
<th>Score</th>
<th>Prog*</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Age</td>
<td></td>
<td>Easyb</td>
<td>.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hardb</td>
<td>.04</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.32</td>
<td>-.33</td>
</tr>
<tr>
<td>B. Sex</td>
<td>Easy</td>
<td>-.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = female</td>
<td>Hard</td>
<td>-.45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 = male</td>
<td>Combined</td>
<td>-.32</td>
<td>.19</td>
<td></td>
</tr>
<tr>
<td>C. Class rank</td>
<td>Easy</td>
<td>.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = freshman</td>
<td>Hard</td>
<td>-.02</td>
<td>.10</td>
<td></td>
</tr>
<tr>
<td>5 = graduate</td>
<td>Combined</td>
<td>-.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td>Easy</td>
<td>-.53*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.24</td>
<td>-.39*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.24</td>
<td>-.39*</td>
<td></td>
</tr>
<tr>
<td>E. Number of programming courses taken</td>
<td>Easy</td>
<td>.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td>Easy</td>
<td>-.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>-.57*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.07</td>
<td>-.15</td>
<td></td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td>Easy</td>
<td>.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.14</td>
<td>-.07</td>
<td></td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td>Easy</td>
<td>.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.16</td>
<td>.25</td>
<td></td>
</tr>
<tr>
<td>I. Years of teaching programming</td>
<td>Easy</td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>-.44</td>
<td>-.51**</td>
<td>.39*</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.51**</td>
<td>.39*</td>
<td></td>
</tr>
<tr>
<td>J. Years of programming</td>
<td>Easy</td>
<td>.38</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>-.24</td>
<td>.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td>Easy</td>
<td>.39</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>-.64*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.30</td>
<td>-.28</td>
<td></td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td>Easy</td>
<td>.37</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.20</td>
<td>-.11</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Program Type</td>
<td>Correlations</td>
<td>Score</td>
<td>Prog</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>--------------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Easy</td>
<td>-.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>-.02</td>
<td>.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Easy</td>
<td>.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>.01</td>
<td>.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O. Years since writing in PL/I</td>
<td>Easy</td>
<td>.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>-.17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>.12</td>
<td>-.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F. Number of programs written</td>
<td>Easy</td>
<td>-.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>-.18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>-.12</td>
<td>.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Easy</td>
<td>.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>-.15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>-.14</td>
<td>.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Easy</td>
<td>-.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>-.49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>.17</td>
<td>-.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Easy</td>
<td>.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>-.69**</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>-.30</td>
<td>.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Easy</td>
<td>.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>.20</td>
<td>-.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Easy</td>
<td>.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>.43</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>.19</td>
<td>-.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Easy</td>
<td>.39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard</td>
<td>-.38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>-.17</td>
<td>.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W. WYLBUR experience</td>
<td>Easy</td>
<td>-.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = low</td>
<td>.29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S = high</td>
<td>.08</td>
<td>.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>.08</td>
<td>.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Correlations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post-Experimental Questionnaire</td>
<td>Program Type</td>
<td>Score</td>
<td>Prog</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Easy</td>
<td>-.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Similarity of program with other programs</td>
<td>Hard</td>
<td>-.09</td>
<td>.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Degree of program understanding</td>
<td>Easy</td>
<td>.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>-.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.45*</td>
<td>-.51**</td>
<td></td>
</tr>
<tr>
<td>3. Ease of program understanding</td>
<td>Easy</td>
<td>-.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>-.70**</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.23</td>
<td>-.48*</td>
<td></td>
</tr>
<tr>
<td>4. Degree to which program is well written</td>
<td>Easy</td>
<td>-.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>-.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.22</td>
<td>-.31</td>
<td></td>
</tr>
</tbody>
</table>

*For purposes of correlation, easy program encoded as 1 and hard program encoded as 2.

° n = 11.

"Coefficient cannot be calculated because survey response is constant for all cases (no one had taught programming).

° n = 7 since three subjects had no experience with Fortran.

° n = 5 since six subjects had no experience with Fortran.

° 1 = low, 5 = high.

° p < .05.

°° p < .01.
APPENDIX Z
Easy Program Version Cloze Test for Experiment 2

1. ---------------*PROCEDURE OPTIONS (MAIN);

/*UPDATE FILE*/
DECLARE 1 TRANS_REC STATIC,
2 TRANS_CODE FIXED(1,0),
2 TRANS_INFO,
3 TRANS_KEY FIXED(4,0),
3 TRANS_AMOUNT FIXED(7,2),
3 TRANS_NAME CHAR(35),
3 TRANS_ADDRESS CHAR(35);

DECLARE 1 OLD_MAST_REC STATIC,
2 OLD_MAST_KEY FIXED(4,0),
2 OLD_MAST_BALANCE FIXED(7,2),
2 OLD_MAST_NAME CHAR(35),
2 OLD_MAST_ADDRESS CHAR(35);

DECLARE 1 NEW_MAST_REC STATIC,
2 NEW_MAST_KEY FIXED(4,0),
2 NEW_MAST_BALANCE FIXED(7,2),
2 NEW_MAST_NAME CHAR(35),
2 NEW_MAST_ADDRESS CHAR(35);

DECLARE CURRENT_KEY STATIC FIXED(4,0);
DECLARE IN_USE STATIC CHAR(5);

2. CALL IN_MAST(TRANS_REC);
3. CALL NEXT_KEY(TRANS_KEY, CURRENT_KEY);
4. CALL PKEY(TRANS_REC, NEW_MAST_REC, CURRENT_KEY);
5. DO WHILE (CURRENT_KEY = 9999);
6. CALL PKEY(OLD_MAST_REC, NEW_MAST_REC, CURRENT_KEY);
7. END;

8. IN_TRAN:PROCEDURE(__________);

/*READ A TRANSACTION RECORD*/
DECLARE 1 TRANS_REC,
2 TRANS_CODE FIXED(1,0),
2 TRANS_INFO,
3 TRANS_KEY FIXED(4,0),
3 TRANS_AMOUNT FIXED(7,2),
3 TRANS_NAME CHAR(*),
3 TRANS_ADDRESS CHAR(*);

DECLARE TRANS_FILE INPUT;

9. ON ENDFILE(TRANS) TRANS = 9999;
10. GET TRANS (TRANS);
11. EDIT (TRANS);
12. (F(1), EDIT(TRANS), TRANS(1),
13. F(5), TRANS(35),
14. RETURN (TRANS(35));
15. END IN_TRAN;
16. PROCEDURE (OLD_MAST_REC); /*READ AN OLD MASTER RECORD*/

DECLARE 1 OLD_MAST_REC,
          2 OLD_MAST_KEY     FIXED(4,0),
          2 OLD_MAST_BALANCE FIXED(7,2),
          2 OLD_MAST_NAME    CHAR(*),
          2 OLD_MAST_ADDRESS CHAR(*);
DECLARE OLD_MAST_FILE "INPUT;

17. ON ENDFILE(____________________) OLD_MAST_KEY =
18. GET FILE(OLDMAST) _______________ ?OLD_MAST_REC__;
19. RETURN;
END IN_MAST;

20. PROCEDURE TRANS.KEY, OLD_MAST_KEY,
    /*CHOOSE NEXT KEY*/
    ____________________________;

DECLARE (TRANS.KEY,OLD_MAST_KEY,CURRENT_KEY) FIXED(4,0);
21. IF OLD_MAST_KEY < = ____________________________;
22. THEN CURRENT_KEY = ____________________________;
23. ELSE CURRENT_KEY = ____________________________;
24. RETURN;
END NEXT_KEY;

25. PROCEDURE (_____________________,OLD_MAST_REC,
    /*PROCESS ONE KEY*/
    _________,CURRENT_KEY);

DECLARE 1 TRANS_REC,
          2 TRANS_CODE     FIXED(1,0),
          2 TRANS_INFO;
          3 TRANS_KEY     FIXED(4,0),
          3 TRANS_AMOUNT  FIXED(7,2),
          3 TRANS_NAME    CHAR(*),
          3 TRANS_ADDRESS CHAR(*);
DECLARE 1 OLD_MAST_REC,
          2 OLD_MAST_KEY    FIXED(4,0),
          2 OLD_MAST_BALANCE FIXED(7,2),
          2 OLD_MAST_NAME   CHAR(*),
          2 OLD_MAST_ADDRESS CHAR(*);
DECLARE 1 NEW_MAST_REC,
          2 NEW_MAST_KEY    FIXED(4,0),
          2 NEW_MAST_BALANCE FIXED(7,2),
          2 NEW_MAST_NAME   CHAR(*),
          2 NEW_MAST_ADDRESS CHAR(*);
DECLARE CURRENT_KEY FIXED(4,0);
DECLARE IN_USE    CHAR(*);

26. CALL INIT(____________________, OLD_MAST_REC, IN_USE, );
27. DO WHILE (TRANS.KEY = ____________________________);
28. CALL PTRANS(TRANS_REC, ___________ , IN_USE);
29. END;
30. CALL FINAL(____________________, IN_USE);
31. CALL NEXT_KEY(_______________, OLD_MAST_KEY, CURRENT_KEY);
32. RETURN;
33. END ______________________;
37. INIT: PROCEDURE (NEW_MAST_REC, CURRENT_KEY, IN_USE); 

/* INITIAL STATUS */

DECLARE 1 OLD_MAST_REC,
  2 OLD_MAST_KEY FIXED(4,0),
  2 OLD_MAST_BALANCE FIXED(7,2),
  2 OLD_MAST_NAME CHAR(*),
  2 OLD_MAST_ADDRESS CHAR(*);
DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME CHAR(*),
  2 NEW_MAST_ADDRESS CHAR(*);
DECLARE CURRENT_KEY FIXED(4,0);
DECLARE IN_USE CHAR(*);

38. IF CURRENT_KEY = OLD_MAST_REC;
39. THEN DO;
40. NEW_MAST_RECfadeOut(OLD_MAST_REC);
41. ELSE IN_USE = 'FALSE';
42. RETURN;
END INIT;

43. ----------------------------------: PROCEDURE (TRANS_REC, NEW_MAST_REC, IN_USE);
44. ----------------------------------

/* PROCESS ONE TRANSACTION */

DECLARE 1 TRANS_REC,
  2 TRANS_CODE FIXED(1,0),
  2 TRANS_INFO,
  3 TRANS_KEY FIXED(4,0),
  3 TRANS_AMOUNT FIXED(7,2),
  3 TRANS_NAME CHAR(*),
  3 TRANS_ADDRESS CHAR(*);
DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME CHAR(*),
  2 NEW_MAST_ADDRESS CHAR(*);
DECLARE IN_USE CHAR(*);

45. CALL MODIFY (TRANS_REC, IN_USE);
46. CALL IN_TRAN (TRANS_REC, IN_USE);
47. RETURN;
END PTRANS;
47. **FINAL:PROCEDURE(------------------------,IN_USE);**
    
    
    DECLARE 1 NEW_MAST_REC,
    2 NEW_MAST_KEY FIXED(4,0),
    2 NEW_MAST_BALANCE FIXED(7,2),
    2 NEW_MAST_NAME CHAR(*),
    2 NEW_MAST_ADDRESS CHAR(*);
    DECLARE IN_USE CHAR(*);
    DECLARE NEW_MAST FILE OUTPUT;
    
    48. **IF IN_USE**
    49. **THEN PUT FILE?------------------------) EDIT(NEW_MAST_REC)
    50. )
    51. )
    52. )
    53. RETURN;
    END FINAL;
    
    54. -------------------:PROCEDURE(TRANS_REC,NEW_MAST_REC,
    55. -------------------)
    
    **/*MODIFY NEW MASTER*/**
    
    DECLARE 1 TRANS_REC,
    2 TRANS_CODE FIXED(1,0),
    2 TRANS_INFO,
    3 TRANS_KEY FIXED(4,0),
    3 TRANS_AMOUNT FIXED(7,2),
    3 TRANS_NAME CHAR(*),
    3 TRANS_ADDRESS CHAR(*);
    DECLARE 1 NEW_MAST_REC,
    2 NEW_MAST_KEY FIXED(4,0),
    2 NEW_MAST_BALANCE FIXED(7,2),
    2 NEW_MAST_NAME CHAR(*),
    2 NEW_MAST_ADDRESS CHAR(*);
    DECLARE IN_USE CHAR(*);
    
    56. IF TRANS_CODE =
    57. THEN CALL PADDD(TRANS_INFO,-----------------2,IN_USE);
    58. ELSE IF TRANS_CODE
    59. THEN CALL PDELETE?-------------------);
    60. ELSE IF TRANS_CODE =
    61. THEN CALL PREPLACE?TRANS_RARE------
        NEW_MAST_NAME,NEW_MAST_ADDRESS;
    62. ELSE IF
    63. THEN "CACC-------------------"(TRANS_AMOUNT,NEW_MAST_BALANCE);
    64. ELSE IF
    65. THEN "CACC-------------------"(TRANS_AMOUNT,NEW_MAST_BALANCE);
    66. RETURN;
    END -------------------;
67. PADD:PROCEDURE (TRANS_INFO, ______________, IN_USE);
   /*PROCESS DIFFERENT MODIFICATIONS*/
   DECLARE 1 TRANS_INFO,
      2 KEY     FIXED (4, 0),
      2 AMOUNT  FIXED (7, 2),
      2 NAME    CHAR (*),
      2 ADDRESS  CHAR (*);
   DECLARE 1 NEW_MAST_REC,
      2 KEY     FIXED (4, 0),
      2 BALANCE FIXED (7, 2),
      2 NAME    CHAR (*),
      2 ADDRESS  CHAR (*);
   DECLARE (IN_USE, TRANS_NAME, TRANS_ADDRESS, NEW_MAST_NAME, NEW_MAST_ADDRESS) CHAR (*);
   DECLARE (TRANS_AMOUNT, NEW_MAST_BALANCE) FIXED (7, 2);
   /*ADDITION*/
   NEW_MAST_REC TRANS_INFO;
   IN_USE = TRUE;
   RETURN;

70. PDELETE:ENTRY (______________________);
   /*DELETION*/
   IN_USE = ______________;
   RETURN;

72. PREPLACE:ENTRY (TRANS_NAME, ______________, NEW_MAST_NAME, NEW_MAST_ADDRESS);
   /*REPLACEMENT*/
    IF TRANS_NAME = ______________ THEN TRANS_NAME;
    IF TRANS_ADDRESS = ______________ THEN TRANS_ADDRESS;
   RETURN;
    __________________:ENTRY (TRANS_AMOUNT, NEW_MAST_BALANCE);
   /*DEBIT*/

78. __________________ = NEW_MAST_BALANCE
    TRANS_AMOUNT;
   RETURN;

80. PCREDIT:ENTRY (______________________, NEW_MAST_BALANCE);
   /*CREDIT*/
81. NEW_MAST_BALANCE __________________ = NEW_MAST_BALANCE;
   RETURN;
END PADD;
END UPDATE;

PLEASE TURN IN THIS BOOKLET AS SOON AS YOU ARE FINISHED
APPENDIX A2
Hard Program Version Cloze Test for Experiment 2

1. ---------------------;PROCEDURE OPTIONS(MAIN);

/*UPDATE FILE*/

DECLARE 1 INFO_REC STATIC;

2. KEY FIXED(4,0);

3. AMOUNT Fixed(7,2);

4. NAME CHAR(35);

5. ADDRESS CHAR(35);

DECLARE END_FLAG STATIC CHAR(5);

CALL BEGIN-DDT;

CALL GET INFO(,END_FLAG);

4. DO WHILE(END_FLAG) = 'FALSE');

5. CALL PRC(,END_FLAG);

END;

CALL END_OUT;

6. ---------------------;PROCEDURE(INFO_REC,END_FLAG);

/*PROCESS ONE RECORD*/

DECLARE 1 INFO_REC.

2. KEY FIXED(4,0);

3. AMOUNT Fixed(7,2);

4. NAME CHAR(*)

5. ADDRESS CHAR(*)

DECLARE END_FLAG CHAR(*)

DECLARE NEW STATIC CHAR(5);

DECLARE CODE STATIC FIXED(1,0);

7. CALL

8. IF NEW = 'TRUE' THEN CALL "READ";

9. CALL

10. IF CODE = 1 THEN CALL "PRC";

11. ELSE IF CODE = 3 THEN CALL "PREP.";

12. ELSE IF CODE = 4 THEN CALL "FDT";

13. ELSE IF CODE = THEN CALL "PREP.";

14. ELSE IF CODE = THEN CALL "FDT";

15. ELSE IF CODE = THEN CALL "PREP.";

16. ELSE IF CODE = THEN CALL "FDT";

17. ELSE IF CODE = THEN CALL "PREP.";

18. CALL (INFO_REC,END_FLAG);

RETURN;

19. END ---------------;

216
BEGIN_IN:PROCEDURE;
/*INPUT MODULE*/
/*INITIALIZE INPUT*/
DECLARE 1 TRANS_REC
  2 TRANS_CODE   STATIC  FIXED(1,0),
  2 TRANS_INFO   FIXED(4,0),
  3 TRANS_KEY    FIXED(7,2),
  3 TRANS_NAME   CHAR(35),
  3 TRANS_ADDRESS CHAR(35);
DECLARE 1 OLD_MAST_REC
  2 OLD_MAST_KEY  STATIC  FIXED(4,0),
  2 OLD_MAST_BALANCE FIXED(7,2),
  2 OLD_MAST_NAME  CHAR(35),
  2 OLD_MAST_ADDRESS CHAR(35);
DECLARE 1 INFO_REC,
  2 KEY        FIXED(4,0),
  2 AMOUNT     FIXED(7,2),
  2 NAME       CHAR(*),
  2 ADDRESS    CHAR(*);
DECLARE CURRENT_KEY  STATIC  FIXED(4,0);
DECLARE CODE         STATIC  FIXED(1,0);
DECLARE TYPE          STATIC  FIXED(1,0);
DECLARE NEW_FLAG      STATIC  CHAR(5);
DECLARE (NEW,END_FLAG) CHAR(*);
DECLARE TRANS_FILE_INPUT;
DECLARE OLD_MAST_FILE_INPUT;

ON ENDFILE(______________) TRANS_KEY =

ON ENDFILE(OLDMAST)

GET

EDIT(______________)

F(1),_________________________

F(5),_________________________

GET

EDIT(OLD_MAST_REC)

F(3),_________________________

F(1),_________________________

= 'FALSE';

RETURN;
35. _____________:ENTRY(INFO_REC,END_FLAG);
36. ON __________________(TRANS) TRANS_KEY
37. ON ENDFILE(______________________) "OCD_MAST_KEY"==9999;
38.             ________________;
39. /*GET ONE RECORD*/
40. IF (OLD_MAST_KEY == ______________-9999) &
41. (TRANS_KEY _______________'TRUE')
42. THEN DO;
43. END_FLAG _______________ 'FALSE';
44. RETURN;
45. END;
46. IF OLD_MAST_KEY_________________________ TRANS_KEY
47. THEN DO;
48. INFO_REC ________________ OLD_MAST_REC;
49. TYPE ____________________ 1;
50. GET FICTET____________________ EDIT(OLD_MAST_REC)
51. F(_________________________131),
52. A(_________________________2),
53. A(_________________________35));
54. END;
55. ELSE DO;
56. = TRANS_INFO;
57. = TRANS_CODE;
58. GET________________________(TRANS)
59. EDITT______________________
60. (F(1),____________________,(1),
61. F(_________________________131),
62. A(_________________________2),
63. A(_________________________35));
64. IF KEY____________________ CURRENT_KEY
65. THEN NEW_FLAG_________________ 'FALSE';
66. ELSE DO;
67. NEW_FLAG ___________________ 'TRUE';
68. CURRENT_KEY_________________ KEY;
69. END;
70. RETURN;
71. PTTYPE:ENTRY(_______________);
72. /*GET PROCESSING TYPE*/
73. CODE ______________;
74. RETURN;_____________;
75. NEWGRP:ENTRY(NEW);
76. /*GET GROUP STATUS*/
77. RETURN;_____________ = NEW_FLAG;
78. END _______________;
BEGIN_OUT:PROCEDURE;
/*OUTPUT MODULE*/
/*INITIALIZE OUTPUT*/
DECLARE 1 INFO REC,
    2 KEY    FIXED(4,0),
    2 AMOUNT FIXED(7,2),
    2 NAME CHAR(*),
    2 ADDRESS CHAR(*);
DECLARE 1 NEW_MAST REC STATIC,
    2 NEW_MAST KEY FIXED(4,0),
    2 NEW_MAST_BALANCE FIXED(7,2),
    2 NEW_MAST_NAME CHAR(35),
    2 NEW_MAST_ADDRESS CHAR(35);
DECLARE (TRANS_NAME,TRANS_ADDRESS) CHAR(*);
DECLARE TRANS_AMOUNT FIXED(7,2);
DECLARE BUFFER STATIC CHAR(5);
DECLARE NEWMAST FILE OUTPUT;
BUFF: --------------------- 'EMPTY';
RETURN;

PADD:ENTRY(______________________);
/*PROCESS ADDITION*/
NEW_MAST_REC = ____________________;
BUFFER = _________________________;
RETURN;

PDELETE:ENTRY;
/*PROCESS DELETION*/
BUFFER ________________________ 'EMPTY';
RETURN;

PREPLACE:ENTRY(____________________,TRANS_ADDRESS);
/*PROCESS REPLACE*/
IF TRANS_NAME THEN NEW_MAST_NAME _______ TRANS_NAME;
IF TRANS_ADDRESS THEN NEW_MAST_ADDRESS _______ TRANS_ADDRESS;
RETURN;

PDEBIT:ENTRY(______________________);
/*PROCESS DEBIT*/
NEW_MAST_BALANCE = __________________ - TRANS_AMOUNT;
RETURN;

______________________:ENTRY(TRANS_AMOUNT);
/*CREDITS*/
NEW_MAST_BALANCE ___________ NEW_MAST_BALANCE
RETURN;
NEW_KEY:ENTRY;
/*PROCESS A NEW KEY*/
02. IF BUFFER ___________________ 'FULL'
    THEN DO:
03.     PUT FILE(_____________(II), EDIT(NEW_MAST_REC)
04.         (FI(_____________),
05.         FF(_____________),
06.         FA(3§T,_____________),(35));
07.     BUFFER ___________________ 'EMPTY';
08. END;
RETURN;

END_OUT:ENTRY;
/*FINALIZE OUTPUT PROCESSING*/
09. IF ＊__________ = 'FULL'
10. THEN PUTT(__________ (NEWMAST)
11.     EDIT(_____________ )
12.     (COL(II),_____________ (3),
13.     F(_____________,
14.     A(_____________),A35));
15. RETURN;
16. END;
END UPDATE;

PLEASE TURN IN THIS BOOKLET AS SOON AS YOU ARE FINISHED
## APPENDIX BB

Means and Standard Deviations of Background Survey Responses and Post-Experimental Questionnaire Responses for Experiment 2

<table>
<thead>
<tr>
<th>Variable</th>
<th>Response</th>
<th>Program Type</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Survey</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. Age</td>
<td></td>
<td>Easy</td>
<td>23.9</td>
<td>5.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>25.6</td>
<td>6.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>24.8</td>
<td>7.0</td>
</tr>
<tr>
<td>B. Sex</td>
<td></td>
<td>Easy</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>1 = female</td>
<td></td>
<td>Hard</td>
<td>1.8</td>
<td>1.4</td>
</tr>
<tr>
<td>2 = male</td>
<td></td>
<td>Combined</td>
<td>1.6</td>
<td>1.5</td>
</tr>
<tr>
<td>C. Class rank</td>
<td></td>
<td>Easy</td>
<td>3.5</td>
<td>0.7</td>
</tr>
<tr>
<td>1 = freshman</td>
<td></td>
<td>Hard</td>
<td>3.6</td>
<td>0.5</td>
</tr>
<tr>
<td>5 = graduate</td>
<td></td>
<td>Combined</td>
<td>3.6</td>
<td>0.6</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td></td>
<td>Easy</td>
<td>3.1</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>3.0</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>3.1</td>
<td>0.4</td>
</tr>
<tr>
<td>E. Number of programming courses taken</td>
<td></td>
<td>Easy</td>
<td>5.9</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>7.2</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>6.5</td>
<td>1.5</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td></td>
<td>Easy</td>
<td>3.4</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>3.2</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>3.3</td>
<td>0.4</td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td></td>
<td>Easy</td>
<td>4.3</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>4.6</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>4.4</td>
<td>2.2</td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td></td>
<td>Easy</td>
<td>.3</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.1</td>
<td>.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.2</td>
<td>.4</td>
</tr>
<tr>
<td>I. Years of teaching programming</td>
<td></td>
<td>Easy</td>
<td>.0</td>
<td>.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>.2</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>.1</td>
<td>.3</td>
</tr>
<tr>
<td>J. Years of programming</td>
<td></td>
<td>Easy</td>
<td>3.3</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>2.7</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>3.0</td>
<td>1.8</td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td></td>
<td>Easy</td>
<td>1.3</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>1.1</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>1.2</td>
<td>1.6</td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td></td>
<td>Easy</td>
<td>1.6</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hard</td>
<td>2.0</td>
<td>.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>1.8</td>
<td>.8</td>
</tr>
<tr>
<td>Variable</td>
<td>Program Type</td>
<td>Mean</td>
<td>Std Dev</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------</td>
<td>------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Easy</td>
<td>0.3</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>0.0</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>0.2</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Easy</td>
<td>2.5</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>1.2</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1.8</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>O. Years since writing in PL/I</td>
<td>Easy</td>
<td>0.3</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>0.0</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>0.1</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Easy</td>
<td>48.5</td>
<td>36.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>51.6</td>
<td>50.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>50.1</td>
<td>43.0</td>
<td></td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Easy</td>
<td>1381.8</td>
<td>1330.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>1089.1</td>
<td>701.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1235.5</td>
<td>1048.9</td>
<td></td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Easy</td>
<td>9.1</td>
<td>11.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>8.2</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>8.6</td>
<td>9.4</td>
<td></td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Easy</td>
<td>152.7</td>
<td>230.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>182.3</td>
<td>186.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>167.5</td>
<td>205.4</td>
<td></td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Easy</td>
<td>23.1</td>
<td>6.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>21.2</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>22.1</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Easy</td>
<td>1381.8</td>
<td>1330.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>1070.9</td>
<td>715.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1226.4</td>
<td>1054.6</td>
<td></td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Easy</td>
<td>4.2</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>4.6</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.4</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>W. WYLBUR experience</td>
<td>Easy</td>
<td>3.6</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>1 = low</td>
<td>Hard</td>
<td>3.8</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>5 = high</td>
<td>Combined</td>
<td>3.7</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Post-Experimental Questionnaire</td>
<td>Response</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program Type</td>
<td>Mean</td>
<td>Std Dev</td>
</tr>
<tr>
<td>1. Similarity of program with other programs</td>
<td>Easy</td>
<td>4.1</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>4.5</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.3</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>2. Degree of program understanding</td>
<td>Easy</td>
<td>3.8</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>4.4</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.1</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>3. Ease of program understanding</td>
<td>Easy</td>
<td>3.5</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>3.9</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.7</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>4. Degree to which program is well written</td>
<td>Easy</td>
<td>3.5</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>4.1</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.8</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

*Except where noted, n = 11.*

*n = 7 since four subjects had no experience with Fortran.*

*n = 8 since three subjects had no experience with Fortran.*

*1 = low, 5 = high.*
**APPENDIX CC**

Correlations of Background Survey Responses and Post-Experimental Questionnaire Responses with Independent and Dependent Variables for Experiment 2

<table>
<thead>
<tr>
<th>Variable</th>
<th>Correlations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background Survey</strong></td>
<td><strong>Program Type</strong></td>
</tr>
<tr>
<td>A. Age</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>B. Sex</td>
<td>Easy</td>
</tr>
<tr>
<td>1 = female</td>
<td>Hard</td>
</tr>
<tr>
<td>2 = male</td>
<td>Combined</td>
</tr>
<tr>
<td>C. Class rank</td>
<td>Easy</td>
</tr>
<tr>
<td>1 = freshman</td>
<td>Hard</td>
</tr>
<tr>
<td>5 = graduate</td>
<td>Combined</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>E. Number of programming courses taken</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>I. Years of teaching programming</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>J. Years of programming</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Variable</td>
<td>Background Survey</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>O. Years since writing in PL/I</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
</tr>
<tr>
<td>W. WYLBUR experience</td>
<td>Easy</td>
</tr>
<tr>
<td>1 = low</td>
<td>Hard</td>
</tr>
<tr>
<td>5 = high</td>
<td>Combined</td>
</tr>
</tbody>
</table>
### Variable Correlations

<table>
<thead>
<tr>
<th>Post-Experimental* Questionnaire</th>
<th>Program Type</th>
<th>Score</th>
<th>Prog</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Similarity of program with other programs</td>
<td>Easy</td>
<td>.72**</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.32</td>
<td>.18</td>
</tr>
<tr>
<td>2. Degree of program understanding</td>
<td>Easy</td>
<td>.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.19</td>
<td>.34</td>
</tr>
<tr>
<td>3. Ease of program understanding</td>
<td>Easy</td>
<td>-.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.04</td>
<td>.21</td>
</tr>
<tr>
<td>4. Degree to which program is well written</td>
<td>Easy</td>
<td>-.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.05</td>
<td>.33</td>
</tr>
</tbody>
</table>

*aFor purposes of correlation, easy program encoded as 1 and hard program encoded as 2.

n = 11.

*bCoefficient cannot be calculated because survey response is constant for all cases (no one had taught programming).

*n = 7 since four subjects had no experience with Fortran.

*n = 8 since three subjects had no experience with Fortran.

1 = low, 5 = high.

*p < .05.

**p < .01.
APPENDIX DD

Listing of Program-Dependent Items In the Cloze Tests
Used in Experiments 1 through 4

Each list of numbers corresponds to the numbered items on the respective cloze tests.

EXPERIMENT 1

Easy program version [see Appendix V]
2, 19, 28, 30, 47, 62, 75, 98, 111, 112, 115, 116, 132, 133, 135

Hard program version [see Appendix W]
2, 6, 7, 8, 18, 19, 21, 22, 24, 49, 51, 52, 53, 54, 55, 56, 58, 59, 67, 68, 86, 87, 90, 91, 93, 96, 97

EXPERIMENT 2

Easy program version [see Appendix I]
5, 9, 12, 13, 18, 20, 25, 26, 27, 32, 38, 48, 50, 51, 56, 58, 59, 60, 61, 62, 63, 64, 65, 71, 72, 73, 74, 75, 76, 77, 78, 79

Hard program version [see Appendix AA]
4, 6, 8, 10, 12, 13, 14, 15, 17, 18, 19, 21, 22, 25, 26, 33, 34, 35, 39, 40, 41, 44, 51, 52, 55, 56, 59, 64, 70, 72, 73, 75, 77, 78, 82, 91

EXPERIMENT 3

Parameter program version [see Appendix GG]
3, 8, 19, 23, 25, 32, 34, 35, 39, 40, 41, 43, 44, 49, 50, 52, 57

Global program version [see Appendix HH]
2, 3, 4, 6, 10, 14, 16, 20, 21, 22, 23, 24, 25, 27, 28, 29, 31, 33, 35, 39, 40, 41, 42, 47, 44, 45, 47, 48, 51, 53, 56, 59

EXPERIMENT 4

Parameter program version [see Appendix Kh]
3, 6, 10, 14, 16, 20, 21, 22, 24, 29, 33, 35, 39, 41, 42, 43, 45, 48, 51, 53

Global program version [see Appendix LL]
2, 3, 8, 19, 23, 24, 25, 32, 34, 35, 39, 40, 41, 42, 43, 44, 49, 50, 51, 52, 53, 56, 57

227
APPENDIX EE

Listing of Parameter Version of Program for Experiments 3 and 4

UPDATE:PROCEDURE OPTIONS(MAIN);
/
UPDATE FILE*/

DECLARE 1 TRANS REC
   2 TRANS CODE FIXED(1,0),
   2 TRANS INFO,
      3 TRANS KEY FIXED(4,0),
      3 TRANS AMOUNT FIXED(7,2),
      3 TRANS NAME CHAR(35),
      3 TRANS ADDRESS CHAR(35);

DECLARE 1 OLD MAST REC
   2 OLD MAST KEY FIXED(4,0),
   2 OLD MAST BALANCE FIXED(7,2),
   2 OLD MAST NAME CHAR(35),
   2 OLD MAST ADDRESS CHAR(35);

DECLARE 1 NEW MAST REC
   2 NEW MAST KEY FIXED(4,0),
   2 NEW MAST BALANCE FIXED(7,2),
   2 NEW MAST NAME CHAR(35),
   2 NEW MAST ADDRESS CHAR(35);

DECLARE CURRENT_KEY FIXED(4,0);
DECLARE IN_USE CHAR(5);

CALL IN_TRAN(TRANS_REC);
CALL IN_MAST(OLD_MAST_REC);
CALL NEXT KEY(TRANS KEY,OLD MAST KEY,CURRENT KEY);
DO WHILE(CURRENT KEY ^= 9999);
   CALL PKEY(TRANS_REC,OLD_MAST_REC,NEW_MAST_REC,IN_USE,CURRENT KEY);
END;

IN_TRAN:PROCEDURE(TRANS_REC);
/
READ A TRANSACTION RECORD*/

DECLARE 1 TRANS REC
   2 TRANS CODE FIXED(1,0),
   2 TRANS INFO,
      3 TRANS KEY FIXED(4,0),
      3 TRANS AMOUNT FIXED(7,2),
      3 TRANS NAME CHAR(*),
      3 TRANS ADDRESS CHAR(*);

DECLARE TRANS FILE INPUT;

ON ENDFILE (TRANS) TRANS KEY = 9999;
GET FILE(TRANS) EDIT(TRANS_REC)
   (F(1),X(1),F(3),F(5,2),A(35),A(35));
RETURN;
END IN_TRAN;
IN_MAST:PROCEDURE(OLD_MAST_REC);

/*READ AN OLD MASTER RECORD*/

DECLARE 1 OLD_MAST_REC,
    2 OLD_MAST_KEY FIXED(4,0),
    2 OLD_MAST_BALANCE FIXED(7,2),
    2 OLD_MAST_NAME CHAR(*),
    2 OLD_MAST_ADDRESS CHAR(*);
DECLARE OLDMAST-FILE-INPUT;

ON ENDFILE(OLDMAST) OLD_MAST_KEY = 9999;
GET FILE(OLDMAST) EDIT(OLD_MAST_REC) (F(3),F(7,2),A(35),A(35));
RETURN;
END IN_MAST;

NEXT_KEY:PROCEDURE(TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY);

/*CHOOSE NEXT KEY*/

DECLARE (TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY) FIXED(4,0);

IF OLD_MAST_KEY <= TRANS_KEY THEN CURRENT_KEY = OLD_MAST_KEY;
ELSE CURRENT_KEY = TRANS_KEY;
RETURN;
END NEXT_KEY;

PKEY:PROCEDURE(TRANS_REC,OLD_MAST_REC,NEW_MAST_REC,IN_USE,CURRENT_KEY);

/*PROCESS ONE KEY*/

DECLARE 1 TRANS_REC,
    2 TRANS_CODE FIXED(1,0),
    2 TRANS_INFO,
    3 TRANS_KEY FIXED(4,0),
    3 TRANS_AMOUNT FIXED(7,2),
    3 TRANS_NAME CHAR(*),
    3 TRANS_ADDRESS CHAR(*);
DECLARE 1 OLD_MAST_REC,
    2 OLD_MAST_KEY FIXED(4,0),
    2 OLD_MAST_BALANCE FIXED(7,2),
    2 OLD_MAST_NAME CHAR(*),
    2 OLD_MAST_ADDRESS CHAR(*);
DECLARE 1 NEW_MAST_REC,
    2 NEW_MAST_KEY FIXED(4,0),
    2 NEW_MAST_BALANCE FIXED(7,2),
    2 NEW_MAST_NAME CHAR(*),
    2 NEW_MAST_ADDRESS CHAR(*);
DECLARE IN_USE CHAR(*);
DECLARE CURRENT_KEY FIXED(4,0);

CALL INIT(NEW_MAST_REC,OLD_MAST_REC,IN_USE,CURRENT_KEY);
DO WHILE(TRANS_KEY = CURRENT_KEY);
    CALL PTRANS(TRANS_REC,NEW_MAST_REC,IN_USE);
END;
CALL FINAL(NEW_MAST_REC,IN_USE);
CALL NEXT_KEY(TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY);
RETURN;
END PKEY;
INIT:PROCEDURE (NEW_MAST_REC, OLD_MAST_REC, IN_USE, CURRENT_KEY);

/*INITIAL STATUS*/

DECLARE 1 OLD_MAST_REC,
  2 OLD_MAST_KEY    FIXED(4,0),
  2 OLD_MAST_BALANCE FIXED(7,2),
  2 OLD_MAST_NAME   CHAR(*),
  2 OLD_MAST_ADDRESS CHAR(*);

DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY     FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME    CHAR(*),
  2 NEW_MAST_ADDRESS CHAR(*);

DECLARE IN_USE    CHAR(*);

DECLARE CURRENT_KEY FIXED(4,0);

IF OLD_MAST_KEY = CURRENT_KEY THEN DO:
  NEW_MAST_REC = OLD_MAST_REC;
  IN USE = 'TRUE';
  CALL IN_MAST(OLD_MAST_REC);
END;
ELSE IN_USE = 'FALSE';
RETURN;
END INIT;

PTRANS:PROCEDURE (TRANS_REC, NEW_MAST_REC, IN_USE);

/*PROCESS ONE TRANSACTION*/

DECLARE 1 TRANS_REC,
  2 TRANS_CODE    FIXED(1,0),
  2 TRANS_INFO,
    3 TRANS_KEY    FIXED(4,0),
    3 TRANS_AMOUNT FIXED(7,2),
    3 TRANS_NAME   CHAR(*),
    3 TRANS_ADDRESS CHAR(*);

DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY     FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME    CHAR(*),
  2 NEW_MAST_ADDRESS CHAR(*);

DECLARE IN_USE    CHAR(*);

CALL MODIFY(TRANS_INFO, NEW_MAST_REC, IN_USE);
CALL IN_TRAN(TRANS_REC);
RETURN;
END PTRANS;

FINAL:PROCEDURE (NEW_MAST_REC, IN_USE);

/*FINAL STATUS*/

DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY     FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME    CHAR(*),
  2 NEW_MAST_ADDRESS CHAR(*);

DECLARE IN_USE    CHAR(*);

DECLARE NEW_MAST FILE OUTPUT;

IF IN_USE = 'TRUE'
  THEN PUT FILE(NEW_MAST) EDIT(NEW_MAST_REC)
       (CDL(1), FT(3), F(7,2), A(35), A(35));
RETURN;
END FINAL;
MODIFY: PROCEDURE (TRANS_INFO, NEW_MAST_REC, IN_USE);

/*MODIFY NEW MASTER*/

DECLARE 1 TRANSINFO,
  2 TRANS_KEY FIXED(4,0),
  2 TRANS_AMOUNT FIXED(7,2),
  2 TRANS_NAME CHAR(*),
  2 TRANS_ADDRESS CHAR(*);

DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME CHAR(*),
  2 NEW_MAST_ADDRESS CHAR(*);

DECLARE IN_USE CHAR(*);

IF TRANS_CODE = 1 THEN CALC_PADDD(TRANS_INFO, NEW_MAST_REC, IN_USE);
ELSE IF TRANS_CODE = 2 THEN CALC_PDELETE(IN_USE);
ELSE IF TRANS_CODE = 3 THEN CALC_PREPLACE(TRANS_NAME, TRANS_ADDRESS, NEW_MAST_NAME, 
                                NEW_MAST_ADDRESS);
ELSE IF TRANS_CODE = 4 THEN CALC_PDEBIT(TRANS_AMOUNT, NEW_MAST_BALANCE);
ELSE IF TRANS_CODE = 5 THEN CALC_PCREDIT(TRANS_AMOUNT, NEW_MAST_BALANCE);
RETURN;
END MODIFY;

PADDD: PROCEDURE (TRANS_INFO, NEW_MAST_REC, IN_USE);

/*ADDITION*/

DECLARE 1 TRANS_INFO,
  2 KEY FIXED(4,0),
  2 AMOUNT FIXED(7,2),
  2 NAME CHAR(*),
  2 ADDRESS CHAR(*);

DECLARE 1 NEW_MAST_REC,
  2 KEY FIXED(4,0),
  2 BALANCE FIXED(7,2),
  2 NAME CHAR(*),
  2 ADDRESS CHAR(*);

DECLARE IN_USE CHAR(*);

NEW_MAST_REC = TRANS_INFO;
IN_USE = 'TRUE';
RETURN;
END PADDD;

PDELETE: PROCEDURE (IN_USE);

/*DELETION*/

DECLARE IN_USE CHAR(*);

IN_USE = 'FALSE';
RETURN;
END PDELETE;
PREPLACE: PROCEDURE (TRANS_NAME, TRANS_ADDRESS, NEW_MAST_NAME, NEW_MAST_ADDRESS);

/*REPLACEMENT*/

DECLARE TRANS_NAME CHAR(*),
TRANS_ADDRESS CHAR(*),
NEW_MAST_NAME CHAR(*),
NEW_MAST_ADDRESS CHAR(*);

IF TRANS_NAME ^= 
THEN NEW_MAST_NAME = TRANS_NAME;
IF TRANS_ADDRESS ^= 
THEN NEW_MAST_ADDRESS = TRANS_ADDRESS;
RETURN;
END PREPLACE;

PDEBIT: PROCEDURE (TRANS_AMOUNT, NEW_MAST_BALANCE);

/*DEBIT*/

DECLARE TRANS_AMOUNT FIXED(7,2),
NEW_MAST_BALANCE FIXED(7,2);

NEW_MAST_BALANCE = NEW_MAST_BALANCE - TRANS_AMOUNT;
RETURN;
END PDEBIT;

PCREDIT: PROCEDURE (TRANS_AMOUNT, NEW_MAST_BALANCE);

/*CREDIT*/

DECLARE TRANS_AMOUNT FIXED(7,2),
NEW_MAST_BALANCE FIXED(7,2);

NEW_MAST_BALANCE = NEW_MAST_BALANCE + TRANS_AMOUNT;
RETURN;
END PCREDIT;
END UPDATE;
APPENDIX FF

Listing of Global Version of Program for Experiments 3 and 4

UPDATE:PROCEDURE OPTIONS(MAIN);

DECLARE 1 TRANS_REC,
  2 TRANS_CODE FIXED(1,0),
  2 TRANS_INFO,
  3 TRANS_KEY FIXED(4,0),
  3 TRANS_AMOUNT FIXED(7,2),
  3 TRANS_NAME CHAR(35),
  3 TRANS_ADDRESS CHAR(35);

DECLARE 1 OLD_MAST_REC,
  2 OLD_MAST_KEY FIXED(4,0),
  2 OLD_MAST_BALANCE FIXED(7,2),
  2 OLD_MAST_NAME CHAR(35),
  2 OLD_MAST_ADDRESS CHAR(35);

DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME CHAR(35),
  2 NEW_MAST_ADDRESS CHAR(35);

DECLARE CURRENT_KEY FIXED(4,0);
DECLARE IN_USE CHAR(5);
DECLARE TRANS_FILE INPUT;
DECLARE OLDMAST_FILE INPUT;
DECLARE NEWMAST_FILE OUTPUT;

CALL IN_TRAN;
CALL IN_MAST;
CALL NEXT_KEY;
DO WHILE (CURRENT_KEY ^= 9999);
  CALL PKEY;
END;

IN_TRAN:PROCEDURE;

/*READ A TRANSACTION RECORD*/

ON ENDFILE(TRANS) TRANS_KEY = 9999;
GET FILE(TRANS) EDIT(TRANS_REC)
  (F(1),F(1),F(3),F(5,2),A(35),A(35));
RETURN;
END IN_TRAN;

IN_MAST:PROCEDURE;

/*READ AN OLD MASTER RECORD*/

ON ENDFILE(OLDMAST) OLD_MAST_KEY = 9999;
GET FILE(OLDMAST) EDIT(OLD_MAST_REC)
  (F(31,F(7,2),A(35),A(35));
RETURN;
END IN_MAST;

233
NEXT_KEY: PROCEDURE;
/* CHOOSE NEXT KEY */
IF OLD_MAST_KEY <= TRANS_KEY
THEN CURRENT_KEY = OLD_MAST_KEY;
ELSE CURRENT_KEY = TRANS_KEY;
RETURN;
END NEXT_KEY;

PKEY: PROCEDURE;
/* PROCESS ONE KEY */
CALL INIT;
DO WHILE (TRANS_KEY = CURRENT_KEY):
   CALL PTRANS;
END;
CALL FINAL;
CALL NEXT_KEY;
RETURN;
END PKEY;

INIT: PROCEDURE;
/* INITIAL STATUS */
IF OLD_MAST_KEY = CURRENT_KEY
THEN DO;
   NEW_MAST_REC = OLD_MAST_REC;
   IN_USE = 'TRUE';
   CALL IN_MAST;
END;
ELSE IN_USE = 'FALSE';
RETURN;
END INIT;

PTRANS: PROCEDURE;
/* PROCESS ONE TRANSACTION */
CALL MODIFY;
CALL IN_TRNS;
RETURN;
END PTRANS;

FINAL: PROCEDURE;
/* FINAL STATUS */
IF IN_USE = 'TRUE'
THEN PUT FILE(NEWMAST) EDIT(NEW_MAST_REC)
   (COL1),F(73),F(7,2),A(35),A(35));
RETURN;
END FINAL;
MODIFY:PROCEDURE;
/*MODIFY NEW MASTER*/
IF TRANS_CODE = 1
THEN CALC PADD;
ELSE IF TRANS_CODE = 2
THEN CALC PDELETE;
ELSE IF TRANS_CODE = 3
THEN CALC PREPLACE;
ELSE IF TRANS_CODE = 4
THEN CALC PD B E I T;
ELSE IF TRANS_CODE = 5
THEN CALC PCREDIT;
RETURN;
END MODIFY;

PADD:PROCEDURE;
/*ADDITION*/
NEW_MAST_REC = TRANS_INFO;
IN_OSE = 'TRUE';
RETURN;
END PADD;

PDELETE:PROCEDURE;
/*DELETION*/
IN_USE = 'FALSE';
RETURN;
END PDELETE;

PREPLACE:PROCEDURE;
/*REPLACEMENT*/
IF TRANS_NAME ^=
THEN NEW_MAST_NAME = TRANS_NAME;
IF TRANS_ADDRESS ^=
THEN NEW_MAST_ADDRESS = TRANS_ADDRESS;
RETURN;
END PREPLACE;

PD B E I T:PROCEDURE;
/*DEBIT*/
NEW_MAST_BALANCE = NEW_MAST_BALANCE - TRANS_AMOUNT;
RETURN;
END PD B E I T;

PCREDIT:PROCEDURE;
/*CREDIT*/
NEW_MAST_BALANCE = NEW_MAST_BALANCE + TRANS_AMOUNT;
RETURN;
END PCREDIT;
END UPDATE;
APPENDIX GG
Parameter Program Cloze Test for Experiment 3

1. UPDATE:PROCEDURE OPTIONS(_________________);
/*UPDATE FILE*/

DECLARE 1 TRANS REC,
  2 TRANS_CODE       FIXED(1,0),
  2 TRANS_INFO;
DECLARE 1 OLD MAST REC,
  2 OLD_MAST_KEY     FIXED(4,0),
  2 OLD_MAST_BALANCE FIXED(7,2),
  2 OLD_MAST_NAME    CHAR(35),
  2 OLD_MAST_ADDRESS CHAR(35);
DECLARE 1 NEW MAST REC,
  2 NEW_MAST_KEY     FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME    CHAR(35),
  2 NEW_MAST_ADDRESS CHAR(35);
DECLARE CURRENT_KEY     FIXED(4,0);
DECLARE IN_USE          CHAR(5);

CALL IN_TRAN(TRANS_REC);
CALL IN_MAST(OLD_MAST_REC);
CALL PKEY(TRANS_REC,OLD_MAST_REC,NEW_MAST_REC,IN_USE,CURRENT_KEY);
END;

IN_TRAN:PROCEDURE(TRANS_REC);
/*READ A TRANSACTION RECORD*/

DECLARE 1 TRANS REC,
  2 TRANS_CODE       FIXED(1,0),
  2 TRANS_INFO;
DECLARE TRANS FILE INPUT;

ON ______________________ (TRANS) TRANS_KEY
GET FILE(_________________) EDIT TRANS RECT----- 9999;

RETURN;
END IN_TRAN;
IN_MAST:PROCEDURE(OLD_MAST_REC);
/*READ AN OLD MASTER RECORD*/
DECLARE OLD_MAST_REC,
  2 OCD_MAST_KEY       FIXED(4,0),
  2 OLD_MAST_BALANCE  FIXED(7,2),
  2 OLD_MAST_NAME     CHAR(*),
  2 OLD_MAST_ADDRESS  CHAR(*);
DECLARE OLDMAST_FILE"INPUT;
ON __________________________(OLDMAST) OLD_MAST_KEY
GET FILE(____________________) EDIT?OCD_MAST_REC---- 9999;
RETURN;
END _________________;

NEXT_KEY:PROCEDURE(TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY);
/*CHOOSE NEXT KEY*/
DECLARE (TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY) FIXED(4,0);
19. IF OLD_MAST_KEY ______________________ TRANS_KEY
20. THEN CURRENT_KEY ____________________ OLD_MAST_KEY;
21. ELSE CURRENT_KEY ____________________ TRANS_KEY;
RETURN;
END NEXT_KEY;
22. _____________________:PROCEDURE(TRANS_REC,OLD_MAST_REC,NEW_MAST_REC,
  IN_USE,CURRENT_KEY);
/*PROCESS ONE KEY*/
DECLARE 1 TRANS_REC,
  2 TRANS_CODE       FIXED(1,0),
  2 TRANS_INFO,
  3 TRANS_KEY       FIXED(4,0),
  3 TRANS_AMOUNT    FIXED(7,2),
  3 TRANS_NAME     CHAR(*),
  3 TRANS_ADDRESS  CHAR(*);
DECLARE 1 OLD_MAST_REC,
  2 OCD_MAST_KEY       FIXED(4,0),
  2 OLD_MAST_BALANCE  FIXED(7,2),
  2 OLD_MAST_NAME     CHAR(*),
  2 OLD_MAST_ADDRESS  CHAR(*);
DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY       FIXED(4,0),
  2 NEW_MAST_BALANCE  FIXED(7,2),
  2 NEW_MAST_NAME     CHAR(*),
  2 NEW_MAST_ADDRESS  CHAR(*);
DECLARE IN_USE     CHAR(*);
DECLARE CURRENT_KEY FIXED(4,0);
CALL INIT(NEW_MAST_REC,OLD_MAST_REC,IN_USE,CURRENT_KEY);
23. DO WHILE(TRANS_KEY CURRENT_KEY);
   CALL PTRANS(TRANS_REC,NEW_MAST_REC,IN_USE);
END;
24. CALL NEXT_KEY?(TRANS_KEY,OLD_MAST_KEY,CURRENT_KEY);
RETURN;
END PKKEY;
INIT:PROCEDURE(NEW_MAST_REC,OLD_MAST_REC,IN_USE,CURRENT_KEY);

/*INITIAL STATUS*/
DECLARE 1 OLD_MAST_REC,
  2 OLD_MAST_KEY    FIXED(4,0);
  2 OLD_MAST_BALANCE FIXED(7,2);
  2 OLD_MAST_NAME   CHAR(*);
  2 OLD_MAST_ADDRESS CHAR(*);
DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY    FIXED(4,0);
  2 NEW_MAST_BALANCE FIXED(7,2);
  2 NEW_MAST_NAME   CHAR(*);
  2 NEW_MAST_ADDRESS CHAR(*);
DECLARE IN_USE    CHAR(*);
DECLARE CURRENT_KEY FIXED(4,0);
25. IF OLD_MAST_KEY = CURRENT_KEY THEN DO;
26.   NEW_MAST_REC = OLD_MAST_REC;
27.   IN_USE = TRUE;
28. ELSE RETURN( = 'FALSE');
29. END;
30. PROCEDURE(TRANS_REC,NEW_MAST_REC,IN_USE);

/*PROCESS ONE TRANSACTION*/
DECLARE 1 TRANS_REC,
  2 TRANS_CODE    FIXED(1,0);
  2 TRANS_INFO,
  3 TRANS_KEY     FIXED(4,0);
  3 TRANS_AMOUNT  FIXED(7,2);
  3 TRANS_NAME    CHAR(*);
  3 TRANS_ADDRESS CHAR(*);
DECLARE 1 NEW_MAST_REC,
  2 NEW_MAST_KEY    FIXED(4,0);
  2 NEW_MAST_BALANCE FIXED(7,2);
  2 NEW_MAST_NAME   CHAR(*);
  2 NEW_MAST_ADDRESS CHAR(*);
DECLARE IN_USE    CHAR(*);
CALL MODIFY(TRANS_INFO,NEW_MAST_REC,IN_USE);
CALL IN_TRAN(TRANS_REC);
RETURN;
31. END;
FINAL: PROCEDURE (NEW_MAST_REC, IN_USE);

/* FINAL STATUS */

DECLARE 1 NEW_MAST_REC;
  2 NEW_MAST_KEY   FIXED(4,0);
  2 NEW_MAST_BALANCE   FIXED(7,2);
  2 NEW_MAST_NAME    CHAR(*);
  2 NEW_MAST_ADDRESS  CHAR(*);
DECLARE IN_USE    CHAR(*);
DECLARE NEW_MAST_FILE_OUTPUT;

32. IF IN_USE 'TRUE'
33. THEN PUT FILE(---------------------) EDIT(NEW_MAST_REC)
34.     (-----------------------------); A(35);
35.     F[7,------------------------]; 735;
36. RETURN;
37. END_FINAL;

38. ----------------------: PROCEDURE (TRANS_INFO, NEW_MAST_REC, IN_USE);

/* MODIFY NEW MASTER */

DECLARE 1 TRANS_INFO;
  2 TRANS_KEY   FIXED(4,0);
  2 TRANS_AMOUNT   FIXED(7,2);
  2 TRANS_NAME   CHAR(*);
  2 TRANS_ADDRESS  CHAR(*);
DECLARE 1 NEW_MAST_REC;
  2 NEW_MAST_KEY   FIXED(4,0);
  2 NEW_MAST_BALANCE   FIXED(7,2);
  2 NEW_MAST_NAME    CHAR(*);
  2 NEW_MAST_ADDRESS  CHAR(*);
DECLARE IN_USE    CHAR(*);

39. IF TRANS_CODE =
    THEN CALL PADD(TRANS_INFO, NEW_MAST_REC, IN_USE);
40. ELSE IF TRANS_CODE = ' ' THEN CALL DELETE(TRANS_INFO, IN_USE);
41. ELSE IF TRANS_CODE = ' 3' THEN CALL CACC(TRANS_NAME, TRANS_ADDRESS,
                                    NEW_MAST_NAME, NEW_MAST_ADDRESS);
42. ELSE IF TRANS_CODE = ' 5' THEN CALL PCREDIT(TRANS_AMOUNT, NEW_MAST_BALANCE);
43. ELSE IF TRANS_CODE = ' 1' THEN CALL PDDEBIT(TRANS_AMOUNT, NEW_MAST_BALANCE);
44. ELSE IF TRANS_CODE = ' 2' THEN CALL PDDEBIT(TRANS_AMOUNT, NEW_MAST_BALANCE);
45. RETURN;
46. END ----------------------;
PADD: PROCEDURE (TRANS_INFO, NEW_MAST_REC, IN_USE);

/* ADDITION */
DECLARE 1 TRANS_INFO,
    2 KEY FIXED (4, 0),
    2 AMOUNT FIXED (7, 2),
    2 NAME CHAR (*),
    2 ADDRESS CHAR (*);
DECLARE 1 NEW_MAST_REC,
    2 KEY FIXED (4, 0),
    2 BALANCE FIXED (7, 2),
    2 NAME CHAR (*),
    2 ADDRESS CHAR (*);
DECLARE IN_USE CHAR (*);

46. NEW_MAST_REC = TRANS_INFO;
47. IN_USE = TRUE;
RETURN;
END PADD;

48. __________________________: PROCEDURE (IN_USE);
/* DELETION */
DECLARE IN_USE CHAR (*);

49. IN_USE = FALSE;
RETURN;
END PDELETE;

PREPLACE: PROCEDURE (TRANS_NAME, TRANS_ADDRESS, NEW_MAST_NAME,
    NEW_MAST_ADDRESS);

/* REPLACEMENT */
DECLARE TRANS_NAME CHAR (*),
    TRANS_ADDRESS CHAR (*),
    NEW_MAST_NAME CHAR (*),
    NEW_MAST_ADDRESS CHAR (*);

50. IF __________________________ THEN TRANS_NAME = TRANS_NAME;
51. IF __________________________ THEN TRANS_ADDRESS = TRANS_ADDRESS;
52. RETURN;
53. END __________________________;

PDEBIT: PROCEDURE (TRANS_AMOUNT, NEW_MAST_BALANCE);

/* DEBIT */
DECLARE TRANS_AMOUNT FIXED (7, 2),
    NEW_MAST_BALANCE FIXED (7, 2);

54. NEW_MAST_BALANCE = NEW_MAST_BALANCE - TRANS_AMOUNT;
55. RETURN;
END PDEBIT;
PCREDIT: PROCEDURE (TRANS_AMOUNT, NEW_MAST_BALANCE);

/* CREDIT */

DECLARE TRANS_AMOUNT         FIXED (7, 2);
NEW_MAST_BALANCE            FIXED (7, 2);

57. ----------------- = NEW_MAST_BALANCE
   ----------------- TRANS_AMOUNT;
58. RETURN;
END PCREDIT;
59. END;
APPENDIX HH

Global Program Close Test for Experiment 3

1. ______________________ PROCEDURE OPTIONS (MAIN):
   /* UPDATE FILE */
   DECLARE 1 TRANS_REC,
   2 TRANS_CODE,           FIXED(1,0),
   2 TRANS_INFO,
   3 TRANS_KEY,             FIXED(4,0),
   3 TRANS_AMOUNT,          FIXED(7,2),
   3 TRANS_NAME,            CHAR(35),
   3 TRANS_ADDRESS,         CHAR(35);
   DECLARE 1 OLD_MAST_REC,
   2 OLD_MAST_KEY,          FIXED(4,0),
   2 OLD_MAST_BALANCE,      FIXED(7,2),
   2 OLD_MAST_NAME,         CHAR(35),
   2 OLD_MAST_ADDRESS,      CHAR(35);
   DECLARE 1 NEW_MAST_REC,
   2 NEW_MAST_KEY,          FIXED(4,0),
   2 NEW_MAST_BALANCE,      FIXED(7,2),
   2 NEW_MAST_NAME,         CHAR(35),
   2 NEW_MAST_ADDRESS,      CHAR(35);
   DECLARE CURRENT_KEY,     FIXED(4,0);
   DECLARE IN_USE,          CHAR(5);
   DECLARE TRANS_FILE INPUT;
   DECLARE DLODMST_FILE INPUT;
   DECLARE NEMASMAT_FILE OUTPUT;

2. CALL ____________
   CALL IN_MAST;
   CALL NEXT_KEY;
3. DO WHILE (_______________ = 9999);
4.   CALL ________________;
   END;

IN_TRANS: PROCEDURE;
   /* READ A TRANSACTION RECORD */
5. ON ENDFILE (_______________) TRANS_KEY =
6. GET_FILE(TRANS) ___________ (TRANS_REC);
7. (F(_______________),X77),
8. (T(_______________),T3),
9. (T(_______________),3),
10. (T(_______________),2),
11. A(_______________),A(35));
12. END ________________;

242
IN_MAST: PROCEDURE;
  /* READ AN OLD MASTER RECORD */
13.  ON ENDFILE( ---------------------- ) OLD_MAST_KEY =
14.  GET FILE(OLDMAST) ---------------------- (?OLD_MAST_REC);
15.  (F(7,---------------------- );A(35),
16.  RETURN; ---------------------- (35));
17.  END IN_MAST;
18.  ---------------------- :PROCEDURE;
19.  /* CHOOSE NEXT KEY */
20.  IF OLD_MAST_KEY <= ----------------------
21.  THEN CURRENT_KEY = ----------------------;
22.  ELSE CURRENT_KEY = ----------------------;
23.  RETURN ; CURRENT_KEY = ----------------------;
24.  END NEXT_KEY;
25.  PKEY: PROCEDURE;
  /* PROCESS ONE KEY */
26.  CALL ;
27.  CALL PTRTRANS;
28.  RETURN ;
29.  END PKEY;
30.  ---------------------- :PROCEDURE;
  /* INITIAL STATUS */
31.  IF OLD_MAST_KEY = ----------------------
32.  THEN DO;
33.  NEW_MAST_REC = ----------------------;
34.  IN USE = ----------------------;
35.  CALL IN_MAST;
36.  CALL IN_USE ---------------------- 'FALSE';
37.  RETURN;
38.  END INIT;
39.  PTRTRANS: PROCEDURE;
  /* PROCESS ONE TRANSACTION */
40.  CALL;
41.  CALL IN Tran;
42.  RETURN;
43.  END PTRTRANS;
32. ---------------:PROCEDURE;
   /*FINAL STATUS*/
33. IF IN USE =
34. THEN PUT FILE N E W M A S T ( NEW_M A S T_REC)
   (C O L ( __________________________ 7 ; 7 3 ),
35. __________________________ (3 5 ) ;
36. AT (__________________________));
37. RETURN;
   END FINAL;
   
   MODIFY:PROCEDURE;
   /*MODIFY N E W M A S T */
39. IF
40. THEN "C A C C "__________ ;
41. ELSE IF TRANS_CODE = ________________ THEN CALL P D E L E T E ;
42. ELSE IF TRANS_CODE = ________________ THEN CALL P R E P L A C E ;
43. ELSE IF TRANS_CODE = ________________ THEN CALL P C R E D I T ;
44. "C A C C "__________ ;
45. ELSE IF TRANS_CODE = ________________ THEN CALL P C R E D I T ;
46. RETURN;
   END MODIFY;
47. ---------------:PROCEDURE;
   /*A D D I T I O N */
48. NEW_M A S T_REC = ______________;
49. IN USE = ______________;
50. RETURN;
   END P A D D ;
51. P D E L E T E :PROCEDURE;
   /*D E L E T I O N */
52. RETURN;__________ = 'F A L S E ' ;
53. END ______________;
   
   P R E P L A C E :PROCEDURE;
   /*R E P L A C E M E N T */
55. IF TRANS_NAME
56. THEN NEW_M A S T_R A N E ______ TRANS_NAME;
57. IF TRANS_A D D R E S S ______ TRANS_A D D R E S S ;
58. THEN NEW_M A S T_A D D R E S S ______ TRANS_A D D R E S S ;
59. RETURN;
   END P R E P L A C E ;
55. ___________________ : PROCEDURE;
    /*DEBIT*/
56. NEW_MAST_BALANCE = __________________ - TRANS_AMOUNT;
    RETURN;
57. END __________________;

58. PCREDIT: PROCEDURE;
    /*CREDIT*/
59. NEW_MAST_BALANCE _______________ NEW_MAST_BALANCE
    RETURN;
    END PCREDIT;
    END UPDATE;

PLEASE TURN IN THIS BOOKLET AS SOON AS YOU ARE FINISHED
## APPENDIX II

Means and Standard Deviations of Background Survey
Responses for Experiment 3

<table>
<thead>
<tr>
<th>Background Survey Variable</th>
<th>Program Type</th>
<th>Response</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Age</td>
<td>Parameters</td>
<td></td>
<td>21.6</td>
<td>.2</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>22.9</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>22.2</td>
<td>.3</td>
</tr>
<tr>
<td>B. Sex</td>
<td>Parameters</td>
<td></td>
<td>1.5</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>1.6</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>1.5</td>
<td>.5</td>
</tr>
<tr>
<td>C. Class rank</td>
<td>Parameters</td>
<td></td>
<td>3.5</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>3.9</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>3.7</td>
<td>.6</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td>Parameters</td>
<td></td>
<td>2.9</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>2.8</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>2.9</td>
<td>.5</td>
</tr>
<tr>
<td>E. Number of programming courses taken</td>
<td>Parameters</td>
<td></td>
<td>3.9</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>4.2</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>4.1</td>
<td>1.6</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td>Parameters</td>
<td></td>
<td>3.3</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>3.0</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>3.2</td>
<td>.6</td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td>Parameters</td>
<td></td>
<td>3.6</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>4.3</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>3.9</td>
<td>1.2</td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td>Parameters</td>
<td></td>
<td>.4</td>
<td>.7</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.9</td>
<td>.0</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.2</td>
<td>.6</td>
</tr>
<tr>
<td>I. Years of teaching programming</td>
<td>Parameters</td>
<td></td>
<td>.1</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.0</td>
<td>.0</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.1</td>
<td>.2</td>
</tr>
<tr>
<td>J. Years of programming</td>
<td>Parameters</td>
<td></td>
<td>2.6</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>2.4</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>2.5</td>
<td>1.4</td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td>Parameters</td>
<td></td>
<td>.5</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.9</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.7</td>
<td>.8</td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td>Parameters</td>
<td></td>
<td>1.0</td>
<td>.8</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>1.3</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>1.1</td>
<td>1.0</td>
</tr>
<tr>
<td>Background Survey Variable</td>
<td>Program Type</td>
<td>Response</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Parameters</td>
<td>.0</td>
<td>.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.0</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.0</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Parameters</td>
<td>1.7</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>2.1</td>
<td>.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1.9</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>O. Years since writing in PL/I</td>
<td>Parameters</td>
<td>.1</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.2</td>
<td>.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.1</td>
<td>.2</td>
<td></td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Parameters</td>
<td>42.6</td>
<td>56.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>30.1</td>
<td>16.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>36.7</td>
<td>41.9</td>
<td></td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Parameters</td>
<td>1178.0</td>
<td>940.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>852.4</td>
<td>597.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1023.8</td>
<td>793.1</td>
<td></td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Parameters</td>
<td>4.6</td>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>6.7</td>
<td>5.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>5.6</td>
<td>4.9</td>
<td></td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Parameters</td>
<td>210.0</td>
<td>314.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>197.2</td>
<td>174.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>203.9</td>
<td>251.0</td>
<td></td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Parameters</td>
<td>12.4</td>
<td>13.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>11.9</td>
<td>10.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>12.2</td>
<td>11.6</td>
<td></td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Parameters</td>
<td>737.0</td>
<td>929.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>630.6</td>
<td>622.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>686.6</td>
<td>779.4</td>
<td></td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Parameters</td>
<td>3.7</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>4.1</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.9</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>W. NYLBUR experience</td>
<td>Parameters</td>
<td>4.1</td>
<td>.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = low</td>
<td>3.9</td>
<td>.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 = high</td>
<td>4.0</td>
<td>.7</td>
<td></td>
</tr>
</tbody>
</table>

*Except where noted, n = 10 for parameters group and n = 9 for globals group (one subject in globals group did not complete a Background Survey. 

*^n = 7 since three subjects had no experience with Fortran.

*^n = 7 since two subjects had no experience with Fortran.
Correlations of Background Survey Responses with Independent and Dependent Variables for Experiment 3

<table>
<thead>
<tr>
<th>Background Survey Variable</th>
<th>Program Type</th>
<th>Score</th>
<th>Time</th>
<th>Prog</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Age</td>
<td>Parameters</td>
<td>-.13</td>
<td>.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.22</td>
<td>-.39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.09</td>
<td>.01</td>
<td>.28</td>
</tr>
<tr>
<td>B. Sex</td>
<td>Parameters</td>
<td>.17</td>
<td>-.20</td>
<td></td>
</tr>
<tr>
<td>1 = female</td>
<td>Globals</td>
<td>.57</td>
<td>.67</td>
<td></td>
</tr>
<tr>
<td>2 = male</td>
<td>Combined</td>
<td>.29</td>
<td>.21</td>
<td>.06</td>
</tr>
<tr>
<td>C. Class rank</td>
<td>Parameters</td>
<td>.21</td>
<td>-.24</td>
<td></td>
</tr>
<tr>
<td>1 = freshman</td>
<td>Globals</td>
<td>.07</td>
<td>-.48</td>
<td></td>
</tr>
<tr>
<td>5 = graduate</td>
<td>Combined</td>
<td>-.13</td>
<td>-.09</td>
<td>.34</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td>Parameters</td>
<td>.48</td>
<td>-.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.05</td>
<td>-.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.29</td>
<td>-.21</td>
<td>-.14</td>
</tr>
<tr>
<td>E. Number of programming courses taken</td>
<td>Parameters</td>
<td>.05</td>
<td>-.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.31</td>
<td>.39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.12</td>
<td>.20</td>
<td>.10</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td>Parameters</td>
<td>.25</td>
<td>-.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.09</td>
<td>-.19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.29</td>
<td>-.20</td>
<td>-.30</td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td>Parameters</td>
<td>.12</td>
<td>-.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.20</td>
<td>-.35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.04</td>
<td>.01</td>
<td>.33</td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td>Parameters</td>
<td>-.11</td>
<td>-.31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>-.28</td>
<td>.c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.10</td>
<td>-.35</td>
<td>-.32</td>
</tr>
<tr>
<td>I. Years of teaching programming</td>
<td>Parameters</td>
<td>.22</td>
<td>-.74**</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>-.09</td>
<td>-.74**</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.24</td>
<td>-.55**</td>
<td>-.22</td>
</tr>
<tr>
<td>J. Years of programming in</td>
<td>Parameters</td>
<td>.07</td>
<td>-.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.42</td>
<td>.46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.17</td>
<td>-.27</td>
<td>-.06</td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td>Parameters</td>
<td>-.41</td>
<td>.48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.34</td>
<td>.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.08</td>
<td>.42</td>
<td>.27</td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td>Parameters</td>
<td>.43</td>
<td>-.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.56</td>
<td>.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.35</td>
<td>.07</td>
<td>.15</td>
</tr>
</tbody>
</table>
### Background Survey

#### M. Years since writing a program
- **Parameters**: 0.14, -0.02
- **Globals**: -0.67*, -0.35
- **Combined**: -0.32, -0.17, 0.02

#### N. Years since writing in Fortran
- **Parameters**: -0.33, -0.7*, -0.06, -0.20
- **Globals**: 0.04, -0.33, 0.18
- **Combined**: 0.04, -0.33, 0.18

#### O. Years since writing in PL/I
- **Parameters**: -0.26, 0.50
- **Globals**: -0.43, 0.23
- **Combined**: -0.43*, 0.35, 0.29

#### P. Number of programs written
- **Parameters**: 0.24, -0.72**
- **Globals**: 0.49, 0.21
- **Combined**: 0.29, -0.46*, -0.15

#### Q. Longest program (lines)
- **Parameters**: 0.11, 0.11
- **Globals**: 0.57, -0.26
- **Combined**: 0.34, -0.14, -0.21

#### R. Number of programs written in Fortran
- **Parameters**: -0.66*, 0.51
- **Globals**: 0.22, 0.09
- **Combined**: -0.20, 0.33, 0.22

#### S. Longest program in Fortran (lines)
- **Parameters**: -0.51, 0.51
- **Globals**: 0.43, 0.02
- **Combined**: -0.12, 0.26, 0.03

#### T. Number of programs written in PL/1
- **Parameters**: 0.38, -0.12
- **Globals**: 0.45, 0.13
- **Combined**: 0.36, -0.03, -0.02

#### U. Longest program in PL/1 (lines)
- **Parameters**: 0.09, 0.23
- **Globals**: 0.39, -0.29
- **Combined**: 0.21, -0.01, -0.07

#### V. Number of programming languages used
- **Parameters**: -0.15, -0.25
- **Globals**: 0.26, 0.46
- **Combined**: 0.00, 0.21, 0.15

#### W. WYLBUR experience
- **Parameters**: 0.47, -0.49
- **Globals**: -0.07, 0.45
- **Combined**: 0.27, -0.18, -0.15

---

**Correlations**

---

*Except where stated, n = 10 for parameters group and n = 9 for globals group (one subject in globals group did not complete a Background Survey).

For purposes of correlation, parameters program encoded as 1 and globals program encoded as 2.

Coefficient cannot be calculated because survey response is constant for all cases (no one had taught programming or had any professional experience).

n = 7 since three subjects had no experience with Fortran.

* n = 7 since two subjects had no experience with Fortran.

P < .05.

** P < .01.
APPENDIX KK
Parameter Program Close Test for Experiment 4

1. ___________________; PROCEDURE OPTIONS(MAIN);

/*UPDATE FILE*/
DECLARE 1 TRANS_REC
  2 TRANS_CODE FIXED(1,0),
  2 TRANS_INFO,
  3 TRANS_KEY FIXED(4,0),
  3 TRANS_AMOUNT FIXED(7,2),
  3 TRANS_NAME CHAR(35),
  3 TRANS_ADDRESS CHAR(35);
DECLARE 1 OLD_MAST_REC
  2 OLD_MAST_KEY FIXED(4,0),
  2 OLD_MAST_BALANCE FIXED(7,2),
  2 OLD_MAST_NAME CHAR(35),
  2 OLD_MAST_ADDRESS CHAR(35);
DECLARE 1 NEW_MAST_REC
  2 NEW_MAST_KEY FIXED(4,0),
  2 NEW_MAST_BALANCE FIXED(7,2),
  2 NEW_MAST_NAME CHAR(35),
  2 NEW_MAST_ADDRESS CHAR(35);
DECLARE CURRENT_KEY FIXED(4,0);
DECLARE IN_USE CHAR(5);

2. CALL (TRANS_PEC);
   CALL IN_MAST TO OLD_MAST_REC);
   CALL NEXT_KEY (TRANS_KEY, OLD_MAST_KEY,CURRENT_KEY);
3. DO WHILE (________________________)
   CALL _________________ (TRANS_REC,OLD_MAST_REC,NEW_MAST_REC,
                           IN_USE,CURRENT_KEY);
   IN_TRAN:PROCEDURE (TRANS_REC);

/*READ A TRANSACTION RECORD*/
DECLARE 1 TRANS_REC
  2 TRANS_CODE FIXED(1,0),
  2 TRANS_INFO,
  3 TRANS_KEY FIXED(4,0),
  3 TRANS_AMOUNT FIXED(7,2),
  3 TRANS_NAME CHAR(*),
  3 TRANS_ADDRESS CHAR(*);
DECLARE TRANS_FILE INPUT;

5. ON ENDFILE (______________________) TRANS_KEY =
6. GET FILE (TRANS) ___________________ TRANS_REC ______;
  7. (F(_____________________________73),
  8. F(_____________________________2),
  9. A(__________________________,A(35));
10. RETURN;
11. END _________________;
12. 250
IN_MAST: PROCEDURE (OLD_MAST_REC);
/* READ AN OLD MASTER RECORD */
DECLARE 1 OLD_MAST_REC,
   2 OLD_MAST_KEY     FIXED(4,0),
   2 OLD_MAST_BALANCE FIXED(7,2),
   2 OLD_MAST_NAME   CHAR(*),
   2 OLD_MAST_ADDRESS CHAR(*);
DECLARE OLDMAST_FILE INPUT;
ON ENDFILE (____________________) OLD_MAST_KEY =
GET_FILE (OLDMAST) ______________ TOOLD_MAST_REC__;
(F(____________________________),A(35),
RETURN; ______________________(35);
END IN_MAST;

________________________: PROCEDURE (TRANS_KEY, OLD_MAST_KEY, CURRENT_KEY);
/* CHOOSE NEXT KEY */
DECLARE (TRANS_KEY, OLD_MAST_KEY, CURRENT_KEY) FIXED(4,0);
IF OLD_MAST_KEY <= __________________________;
THEN CURRENT_KEY = ______________________;
ELSE CURRENT_KEY = ______________________;
RETURN;
END NEXT_KEY;

PKKEY: PROCEDURE (TRANS_REC, OLD_MAST_REC, NEW_MAST_REC, IN_USE, CURRENT_KEY);
/* PROCESS ONE KEY */
DECLARE 1 TRANS_REC,
   2 TRANS_CODE        FIXED(1,0),
   2 TRANS_INFO,      
   3 TRANS_KEY        FIXED(4,0),
   3 TRANS_AMOUNT     FIXED(7,2),
   3 TRANS_NAME       CHAR(*),
   3 TRANS_ADDRESS    CHAR(*);
DECLARE 1 OLD_MAST_REC,
   2 OLD_MAST_KEY     FIXED(4,0),
   2 OLD_MAST_BALANCE FIXED(7,2),
   2 OLD_MAST_NAME   CHAR(*),
   2 OLD_MAST_ADDRESS CHAR(*);
DECLARE 1 NEW_MAST_REC,
   2 NEW_MAST_KEY     FIXED(4,0),
   2 NEW_MAST_BALANCE FIXED(7,2),
   2 NEW_MAST_NAME   CHAR(*),
   2 NEW_MAST_ADDRESS CHAR(*);
DECLARE IN_USE   CHAR(*);
DECLARE CURRENT_KEY FIXED(4,0);
CALL ________________ (NEW_MAST_REC, OLD_MAST_REC, IN_USE, CURRENT_KEY);
DO WHILE (TRANS_KEY =
CALL PTRANS (TRANS_REC, NEW_MAST_REC, IN_USE);
END;
CALL FINAL (NEW_MAST_REC, IN_USE);
CALL ________________ (TRANS_KEY, OLD_MAST_KEY, CURRENT_KEY);
RETURN; ________________ (TRANS_KEY, OLD_MAST_KEY, CURRENT_KEY);
END PKKEY;
PROCEDURE(NEW_MAST_REC, OLD_MAST_REC, IN_USE, CURRENT_KEY);

/ INITIAL STATUS /

DECLARE 1 OLD_MAST_REC,
   2 OLD_MAST_KEY FIXED(4,0),
   2 OLD_MAST_BALANCE FIXED(7,2),
   2 OLD_MAST_NAME CHAR(*),
   2 OLD_MAST_ADDRESS CHAR(*);

DECLARE 1 NEW_MAST_REC,
   2 NEW_MAST_KEY FIXED(4,0),
   2 NEW_MAST_BALANCE FIXED(7,2),
   2 NEW_MAST_NAME CHAR(*),
   2 NEW_MAST_ADDRESS CHAR(*);

DECLARE IN_USE CHAR(*)
DECLARE CURRENT_KEY FIXED(4,0);

IF OLD_MAST_KEY = THEN DO;
   NEW_MAST_REC = IN_OUSE = ;
   CALL IN_MAST(OLD_MAST_REC);
END;

ELSE IN_USE = 'FALSE';
RETURN;
END INIT;

PTRANS:PROCEDURE(TRANS_REC, NEW_MAST_REC, IN_USE);

/ PROCESS ONE TRANSACTION /

DECLARE 1 TRANS_REC,
   2 TRANS_CODE FIXED(1,0),
   2 TRANS_INFO,
   3 TRANS_KEY FIXED(4,0),
   3 TRANS_AMOUNT FIXED(7,2),
   3 TRANS_NAME CHAR(*),
   3 TRANS_ADDRESS CHAR(*);

DECLARE 1 NEW_MAST_REC,
   2 NEW_MAST_KEY FIXED(4,0),
   2 NEW_MAST_BALANCE FIXED(7,2),
   2 NEW_MAST_NAME CHAR(*),
   2 NEW_MAST_ADDRESS CHAR(*);

DECLARE IN_USE CHAR(*);

CALL _IN Tran TRAN_S_REC ;
RETURN;
END PTRANS;
32. ________________ : PROCEDURE (NEW_MAST_REC, IN_USE);

    /* FINAL STATUS */

    DECLARE 1 NEW_MAST_REC,
    2 NEW_MAST_KEY      FIXED (4, 0),
    2 NEW_MAST_BALANCE  FIXED (7, 2),
    2 NEW_MAST_NAME    CHAR (*),
    2 NEW_MAST_ADDRESS CHAR (*);

    DECLARE IN_USE    CHAR (*);

    DECLARE NEWMAST FILE OUTPUT;

33. IF IN_USE =
34. THEN PUT FILE (NEWMAST) ------------------------------- (NEW_MAST_REC)
35. (COL (__________________________TIFT3),
36. ____________________________T72),
37. ______________________________ (35),
38. AT ________________);

    RETURN;
    END FINAL;

MODIFY: PROCEDURE (TRANS_INFO, NEW_MAST_REC, IN_USE);

    /* MODIFY NEW MASTER */

    DECLARE 1 TRANS_INFO,
    2 TRANS_KEY      FIXED (4, 0),
    2 TRANS_AMOUNT   FIXED (7, 2),
    2 TRANS_NAME    CHAR (*),
    2 TRANS_ADDRESS CHAR (*);

    DECLARE 1 NEW_MAST_REC,
    2 NEW_MAST_KEY      FIXED (4, 0),
    2 NEW_MAST_BALANCE  FIXED (7, 2),
    2 NEW_MAST_NAME    CHAR (*),
    2 NEW_MAST_ADDRESS CHAR (*);

    DECLARE IN_USE    CHAR (*);

39. IF TACC = 1
40. THEN PDELETETIN_USET;----------------------

41. ELSE IF TRANS_CODE = 3
42. THEN PREPACETTRANS_NAME,TRANS_ADDRESS,NEW_MAST_NAME,
        NEW_MAST_ADDRESS);

43. ELSE IF TACC = 4
44. THEN PCREDITTRANS_AMOUNT,NEW_MAST_BALANCE);

        END MODIFY;
46. PROCEDURE (TRANS_INFO, NEW_MAST_REC, IN_USE);
/* ADDITION */
DECLARE 1 TRANS_INFO,
2 KEY     FIXED (4, 0),
2 AMOUNT  FIXED (7, 2),
2 NAME    CHAR (*),
2 ADDRESS CHAR (*);
DECLARE 1 NEW_MAST_REC,
2 KEY     FIXED (4, 0),
2 BALANCE FIXED (7, 2),
2 NAME    CHAR (*),
2 ADDRESS CHAR (*);
DECLARE IN_USE CHAR (*);

47. NEW_MAST_REC =
48. IN_USE =
49. RETURN;
50. END ADD;

PDELETE: PROCEDURE (IN_USE);
/* DELETION */
DECLARE IN_USE CHAR (*);

51. IF TRANS_NAME
52. THEN NEW_MAST_NAME ' ' TRANS_NAME;
53. IF TRANS_ADDRESS
54. THEN NEW_MAST_ADDRESS ' ' TRANS_ADDRESS;
55. RETURN;
56. END REPLACE;

PREPLACE: PROCEDURE (TRANS_NAME, TRANS_ADDRESS, NEW_MAST_NAME,
NEW_MAST_ADDRESS);
/* REPLACEMENT */
DECLARE TRANS_NAME CHAR (*),
TRANS_ADDRESS CHAR (*),
NEW_MAST_NAME CHAR (*),
NEW_MAST_ADDRESS CHAR (*);

57. PROCEDURE (TRANS_AMOUNT, NEW_MAST_BALANCE);
/* DEBIT */
DECLARE TRANS_AMOUNT FIXED (7, 2),
NEW_MAST_BALANCE FIXED (7, 2);

58. NEW_MAST_BALANCE = TRANS_AMOUNT;
59. RETURN;
60. END;
PCREDIT: PROCEDURE (TRANS_AMOUNT, NEW_MAST_BALANCE);

/* CREDIT */

DECLARE TRANS_AMOUNT FIXED (7, 2),
    NEW_MAST_BALANCE FIXED (7, 2);

58. NEW_MAST_BALANCE ------------  NEW_MAST_BALANCE
59.    +-----------------------
RETURN;
END PCREDIT;
END UPDATE;

PLEASE TURN IN THIS BOOKLET AS SOON AS YOU ARE FINISHED
APPENDIX LL

Global Program Cloze Test for Experiment 4

1. UPDATE:PROCEDURE OPTIONS(______________);

/*UPDATE FILE*/

DECLARE 1 TRANS_REC,
    2 TRANS_CODE         FIXED(1,0),
    2 TRANS_INFO,
        3 TRANS_KEY       FIXED(4,0),
        3 TRANS_AMOUNT    FIXED(7,2),
        3 TRANS_NAME      CHAR(35),
        3 TRANS_ADDRESS   CHAR(35);

DECLARE 1 OLD_MAST_REC,
    2 OLD_MAST_KEY      FIXED(4,0),
    2 OLD_MAST_BALANCE  FIXED(7,2),
    2 OLD_MAST_NAME     CHAR(35),
    2 OLD_MAST_ADDRESS  CHAR(35);

DECLARE 1 NEW_MAST_REC,
    2 NEW_MAST_KEY      FIXED(4,0),
    2 NEW_MAST_BALANCE  FIXED(7,2),
    2 NEW_MAST_NAME     CHAR(35),
    2 NEW_MAST_ADDRESS  CHAR(35);

DECLARE CURRENT_KEY       FIXED(4,0);
DECLARE IN_USE             CHAR(5);
DECLARE TRANS_FILE INPUT;
DECLARE OLDMAST_FILE INPUT;
DECLARE NEWMAST FILE OUTPUT;

CALL IN_TRAN;
CALL IN_MAST;

2. CALL

3. DO WHIL( CURRENT_KEY''==''-1)
    CALL PKEY;
END;

IN_TRAN:PROCEDURE;

/READ A TRANSACTION RECORD*/

4. ON ------------------------(TRANS) TRANS_KEY

5. GET FILE(----------------------'EDITTRANSRECT'--- 9999;

6. (-----------------------------(1),

7. '1'-----------------------------(3),

8. 'r'-----------------------------(75,2),

9. 'k'-----------------------------(35),

10. AT-----------------------------);

11. RETURN;
END IN_TRAN;
IN_MAST: PROCEDURE;

/* READ AN OLD MASTER RECORD */

ON __________________________(OLDMAST) OLD_MAST_KEY

GET FILE(_____________________)EDITTED_MAST_REC

RETURN;

NEXT_KEY: PROCEDURE;

/* CHOOSE NEXT KEY */

IF OLD_MAST_KEY ______________________________TRANS_KEY

THEN CURRENT_KEY _________________________OLD_MAST_KEY;

ELSE CURRENT_KEY _________________________TRANS_KEY;

RETURN;

END NEXT_KEY;

PROCEDURE;

/* PROCESS ONE KEY */

CALL INIT;

DO WHILE(TRANS_KEY _________________________CURRENT_KEY);

CALL PTRANS;

END;

CALL NEXT_KEY;

RETURN;

END PKEY;

INIT: PROCEDURE;

/* INITIAL STATUS */

IF OLD_MAST_KEY ______________________________CURRENT_KEY

THEN DO;

NEW_MAST_REC _________________________OLD_MAST_REC;

IN DSE _________________________TRUE;

CALL IN_MAST;

END;

ELSE

RETURN _________________________ = "FALSE";

END

PROCEDURE;

/* PROCESS ONE TRANSACTION */

CALL MODIFY;

CALL IN_TRAN;

RETURN;

END;
FINAL: PROCEDURE;
/*FINAL STATUS*/

32. IF IN USE
33. THEN PUT FILE(----------------- 'TRUE') EDIT(NEW_MAST_REC)
34. (1),
35. F(71,-----------------'), A(35),
36. RETURN;
37. END FINAL;

38. -------------------:PROCEDURE;
/*MODIFY NEW MASTER*/

39. IF TRANS_CODE = -------------------
40. THEN CALL PADD;
41. ELSE IF TRANS_CODE = 2
42. THEN CALL PDELETE;
43. ELSE IF TRANS_CODE = 3
44. THEN CALL PCREDIT;
45. ELSE IF TRANS_CODE = 5
46. THEN CALL PDEBIT;
47. RETURN;
48. END -------------------;

PADD:PROCEDURE;
/*ADDITION*/

46. NEW_MAST_REC ------------------ TRANS_INFO;
47. IN_USE ----------------- 'TRUE';
48. RETURN;
49. END PADD;

48. ------------------:PROCEDURE;
/*DELETION*/

50. IN_USE = ------------------;
51. RETURN;
52. END PDELETE;

PREPLACE:PROCEDURE;
/*REPLACEMENT*/

50. IF ------------------ ^ = TRANS_NAME;
51. THEN ------------------ = TRANS_NAME;
52. IF ------------------ = TRANS_ADDRESS;
53. RETURN;
54. END ------------------;
```
PDEBIT: PROCEDURE;
/*DEBIT*/
55. NEW_MAST_BALANCE -------------- NEW_MAST_BALANCE -
56. RETURN;
    END PDEBIT;

PCREDIT: PROCEDURE;
/*CREDIT*/
57. --------------------- = NEW_MAST_BALANCE
58. --------------------- TRANS_AMOUNT;
59. RETURN;
    END PCREDIT;
60. END ---------------------;

PLEASE TURN IN THIS BOOKLET AS SOON AS YOU ARE FINISHED
```
## APPENDIX MM

### Means and Standard Deviations of Background Survey Responses for Experiment 4

<table>
<thead>
<tr>
<th>Background Survey Variable</th>
<th>Program Type</th>
<th>Parameters</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Age</td>
<td></td>
<td></td>
<td>23.9</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Globals</td>
<td>25.3</td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td>24.6</td>
<td>6.4</td>
</tr>
<tr>
<td>B. Sex</td>
<td></td>
<td></td>
<td>1.7</td>
<td>5.0</td>
</tr>
<tr>
<td>1 = female</td>
<td></td>
<td></td>
<td>1.8</td>
<td>5.0</td>
</tr>
<tr>
<td>2 = male</td>
<td></td>
<td></td>
<td>1.8</td>
<td>5.0</td>
</tr>
<tr>
<td>C. Class rank</td>
<td></td>
<td></td>
<td>4.0</td>
<td>5.0</td>
</tr>
<tr>
<td>1 = freshman</td>
<td></td>
<td></td>
<td>4.0</td>
<td>7.0</td>
</tr>
<tr>
<td>5 = graduate</td>
<td></td>
<td></td>
<td>4.0</td>
<td>7.0</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td></td>
<td></td>
<td>2.8</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.0</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.9</td>
<td>6.6</td>
</tr>
<tr>
<td>E. Number of programming courses taken</td>
<td></td>
<td>Parameters</td>
<td>4.5</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.4</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.4</td>
<td>1.6</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td></td>
<td></td>
<td>3.1</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.3</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.2</td>
<td>1.6</td>
</tr>
<tr>
<td>G. Years of post high school education</td>
<td></td>
<td>Parameters</td>
<td>5.3</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.4</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.9</td>
<td>2.9</td>
</tr>
<tr>
<td>H. Years of professional programming</td>
<td></td>
<td>Parameters</td>
<td>.0</td>
<td>.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.6</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.3</td>
<td>4.9</td>
</tr>
<tr>
<td>I. Years of teaching programming</td>
<td></td>
<td>Parameters</td>
<td>.0</td>
<td>.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.2</td>
<td>.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.1</td>
<td>.0</td>
</tr>
<tr>
<td>J. Years of programming</td>
<td></td>
<td></td>
<td>1.9</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.8</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.9</td>
<td>4.6</td>
</tr>
<tr>
<td>K. Years of programming in Fortran</td>
<td></td>
<td>Parameters</td>
<td>.4</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.0</td>
<td>6.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.7</td>
<td>4.8</td>
</tr>
<tr>
<td>L. Years of programming in PL/I</td>
<td></td>
<td>Parameters</td>
<td>.8</td>
<td>.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.3</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.6</td>
<td>3.1</td>
</tr>
<tr>
<td>Background Survey Variable</td>
<td>Program Type</td>
<td>Mean</td>
<td>Std Dev</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------</td>
<td>-------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>M. Years since writing a program</td>
<td>Parameters</td>
<td>.0</td>
<td>.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.1</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.0</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Parameters $^b$</td>
<td>2.3</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>2.0</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>2.1</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>D. Years since writing in PL/I</td>
<td>Parameters</td>
<td>.1</td>
<td>.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.7</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.4</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Parameters</td>
<td>26.0</td>
<td>8.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>74.8</td>
<td>149.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>50.4</td>
<td>106.3</td>
<td></td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Parameters</td>
<td>941.4</td>
<td>912.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>2425.0</td>
<td>2551.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1683.2</td>
<td>2013.9</td>
<td></td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Parameters</td>
<td>3.6</td>
<td>3.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>45.2</td>
<td>124.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>25.4</td>
<td>88.2</td>
<td></td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Parameters</td>
<td>238.2</td>
<td>186.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>568.0</td>
<td>639.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>403.1</td>
<td>488.6</td>
<td></td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Parameters</td>
<td>11.6</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>9.9</td>
<td>6.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>10.8</td>
<td>7.9</td>
<td></td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Parameters</td>
<td>537.9</td>
<td>341.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>1383.0</td>
<td>2713.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>960.4</td>
<td>1931.5</td>
<td></td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Parameters</td>
<td>4.2</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>4.1</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>4.2</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>W. WYLBUR experience 1 = low</td>
<td>Parameters</td>
<td>4.3</td>
<td>.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>3.4</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>3.8</td>
<td>1.2</td>
<td></td>
</tr>
</tbody>
</table>

$^a$Except where noted, $n = 10$.

$^b$n = 8 since two subjects had no experience with Fortran.

$^c$n = 9 since one subject had no experience with Fortran.
APPENDIX NN

Correlations of Background Survey Responses with Independent and Dependent Variables for Experiment 4

<table>
<thead>
<tr>
<th>Background Survey Variable</th>
<th>Program Type</th>
<th>Score</th>
<th>Time</th>
<th>Prog*</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Age</td>
<td>Parameters</td>
<td>-.24</td>
<td>.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.19</td>
<td>.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.07</td>
<td>.26</td>
<td>.10</td>
</tr>
<tr>
<td>B. Sex</td>
<td>Parameters</td>
<td>.10</td>
<td>-.16</td>
<td></td>
</tr>
<tr>
<td>1 = female</td>
<td>Globals</td>
<td>.19</td>
<td>.77</td>
<td>**</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.15</td>
<td>.31</td>
<td>.12</td>
</tr>
<tr>
<td>C. Class rank</td>
<td>Parameters</td>
<td>.30</td>
<td>.08</td>
<td></td>
</tr>
<tr>
<td>1 = freshman</td>
<td>Globals</td>
<td>.29</td>
<td>.35</td>
<td></td>
</tr>
<tr>
<td>5 = graduate</td>
<td>Combined</td>
<td>.29</td>
<td>.26</td>
<td>.00</td>
</tr>
<tr>
<td>D. Overall GPA</td>
<td>Parameters</td>
<td>-.38</td>
<td>.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.17</td>
<td>-.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.08</td>
<td>-.00</td>
<td>.12</td>
</tr>
<tr>
<td>E. Number of programming</td>
<td>Parameters</td>
<td>.43</td>
<td>.16</td>
<td></td>
</tr>
<tr>
<td>courses taken</td>
<td>Globals</td>
<td>.25</td>
<td>-.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.32</td>
<td>-.05</td>
<td>-.03</td>
</tr>
<tr>
<td>F. Programming courses GPA</td>
<td>Parameters</td>
<td>-.36</td>
<td>-.21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.17</td>
<td>-.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.09</td>
<td>-.25</td>
<td>.17</td>
</tr>
<tr>
<td>G. Years of post high</td>
<td>Parameters</td>
<td>-.70</td>
<td>.34</td>
<td></td>
</tr>
<tr>
<td>school education</td>
<td>Globals</td>
<td>-.22</td>
<td>.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>-.51</td>
<td>-.22</td>
<td>-.16</td>
</tr>
<tr>
<td>H. Years of professional</td>
<td>Parameters</td>
<td>.28</td>
<td>-.37</td>
<td></td>
</tr>
<tr>
<td>programming</td>
<td>Globals</td>
<td>.24</td>
<td>.24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.20</td>
<td>.16</td>
<td>.26</td>
</tr>
<tr>
<td>I. Years of teaching</td>
<td>Parameters</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>programming</td>
<td>Globals</td>
<td>.24</td>
<td>.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.19</td>
<td>.19</td>
<td>.23</td>
</tr>
<tr>
<td>J. Years of programming</td>
<td>Parameters</td>
<td>.03</td>
<td>.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td>.27</td>
<td>.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.22</td>
<td>.23</td>
<td>.22</td>
</tr>
<tr>
<td>K. Years of programming</td>
<td>Parameters</td>
<td>.06</td>
<td>-.40</td>
<td></td>
</tr>
<tr>
<td>in Fortran</td>
<td>Globals</td>
<td>.25</td>
<td>.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.21</td>
<td>.18</td>
<td>.27</td>
</tr>
<tr>
<td>L. Years of programming</td>
<td>Parameters</td>
<td>.31</td>
<td>.25</td>
<td></td>
</tr>
<tr>
<td>in PL/I</td>
<td>Globals</td>
<td>.29</td>
<td>.21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>.26</td>
<td>.16</td>
<td>.25</td>
</tr>
</tbody>
</table>
### Background Survey Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Program Type</th>
<th>Score</th>
<th>Time</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>M. Years since writing a program</td>
<td>Parameters</td>
<td>.10</td>
<td>.02</td>
<td>.59*</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.04</td>
<td>.42*</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.68</td>
<td>.32</td>
</tr>
<tr>
<td>N. Years since writing in Fortran</td>
<td>Parameters</td>
<td>.05</td>
<td>.56</td>
<td>.20</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>-.39</td>
<td>-.12</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>-.36</td>
<td>-.06</td>
</tr>
<tr>
<td>O. Years since writing in PL/I</td>
<td>Parameters</td>
<td>-.17</td>
<td>.39</td>
<td>.44</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.24</td>
<td>.31</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.29</td>
<td>.23</td>
</tr>
<tr>
<td>P. Number of programs written</td>
<td>Parameters</td>
<td>.28</td>
<td>.26</td>
<td>.68*</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.22</td>
<td>.25</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.22</td>
<td>.24</td>
</tr>
<tr>
<td>Q. Longest program (lines)</td>
<td>Parameters</td>
<td>-.43</td>
<td>-.32</td>
<td>.00</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.15</td>
<td>-.17</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.15</td>
<td>.38*</td>
</tr>
<tr>
<td>R. Number of programs written in Fortran</td>
<td>Parameters</td>
<td>.06</td>
<td>.24</td>
<td>.47</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.19</td>
<td>.21</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.19</td>
<td>.23</td>
</tr>
<tr>
<td>S. Longest program in Fortran (lines)</td>
<td>Parameters</td>
<td>-.28</td>
<td>.32</td>
<td>-.16</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.32</td>
<td>.28</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.32</td>
<td>.35</td>
</tr>
<tr>
<td>T. Number of programs written in PL/I</td>
<td>Parameters</td>
<td>.33</td>
<td>.55*</td>
<td>.62*</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.43*</td>
<td>.23</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.43*</td>
<td>.11</td>
</tr>
<tr>
<td>U. Longest program in PL/I (lines)</td>
<td>Parameters</td>
<td>-.30</td>
<td>.54</td>
<td>.15</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.15</td>
<td>.31</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.15</td>
<td>.22</td>
</tr>
<tr>
<td>V. Number of programming languages used</td>
<td>Parameters</td>
<td>.07</td>
<td>.58</td>
<td>.34</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>.32</td>
<td>.44*</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>.32</td>
<td>.03</td>
</tr>
<tr>
<td>W. WYLBUR experience 1 = low 5 = high</td>
<td>Parameters</td>
<td>-.29</td>
<td>.55*</td>
<td>-.45</td>
</tr>
<tr>
<td></td>
<td>Globals</td>
<td></td>
<td>-.45</td>
<td>-.59*</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td></td>
<td>-.45</td>
<td>-.39*</td>
</tr>
</tbody>
</table>

*For purposes of correlation, parameters program encoded as 1 and globals program encoded as 2.*

*Except where stated, n = 10.*

*Coefficient cannot be calculated because survey response is constant for all cases (no one had taught programming).*

*n = 8 since two subjects had no experience with Fortran.*

*n = 9 since one subject had no experience with Fortran.*

*P < .05.*

**P < .01.*