Crosby, Martha Elizabeth

NATURAL VERSUS COMPUTER LANGUAGES: A READING COMPARISON

University of Hawaii PH.D. 1986

NATURAL VERSUS COMPUTER LANGUAGES: A READING COMPARISON

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAII IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN EDUCATIONAL PSYCHOLOGY

MAY 1986

By

Martha Elizabeth Crosby

Dissertation Committee:

Peter Dunn-Rankin, Chairman
Mary E. Brandt
Morris K. Lai
Ann M. Peters
W. Wesley Peterson

ACKNOWLEDGEMENTS

I wish to thank my committee for their advise and encouragement. I would especially like to thank my advisor Dr. Peter Dunn-Rankin for his guidance, particularly in the area of reading. I would also like to thank Dr. W. Wesley Peterson for suggesting this area of research, for designing the computers used in the experiments, and for allowing me to use the facilities of the Computer Science department. In addition, I would like to thank Dr. Mary E. Brandt, Dr. Ann M. Peters, and Dr. Morris K. Lai for their careful editing and helpful remarks. I am also grateful to my friend, Dr. Esmè Hoban, for helping me keep the effort involved in producing a dissertation in perspective. Finally, I would like to express my gratitude to my husband, Peter, for his unceasing patience and support for this project.

ABSTRACT

This study investigated how non-expert computer programmers read and remembered computer programs. Two experiments were conducted. The first experiment questioned whether or not the subject's knowledge of the program's purpose interacted with features of the text to influence program recall. The second experiment compared the subject's ability to comprehend and remember both standard English and Pascal versions of the same algorithm.

Experiment one investigated the influence on program recall of the subject's knowledge of the purpose or gist of the computer program, the explicitness or transparency of the algorithm, and the propositional density. For this experiment thirty-two second semester computer science students read short, six to twelve line, programs. The independent variables were whether or not the subject knew the program's gist, low and high propositional density, and explicit and implicit algorithms. The dependent variable was whether or not the subject recalled 95% of the program. It was found that knowledge of the program's purpose facilitated program recall. However, if the subjects were not aware of the program's gist, features of the program text influenced program recall. When the program's gist was unknown, the least terse programs (those with the low

propositional density) and least transparent programs (those with the implicit algorithms) were recalled more sucessfully.

Experiment two investigated comprehension of programs written in a natural programming language (English) and a computer programming language (Pascal). In addition to the language type, independent variables which represented features of the text included program size, density, and the depth of propositional hierarchy. Attributes of the subject which were used as independent variables included the subject's knowledge of the program's gist, training, language ability, and programming experience. The dependent variables were recognition memory scores, subject's knowledge of the program's gist, cloze scores, number of seconds to read the program, and average number of seconds to correctly answer questions about the program.

Variables which were shown to interact with language type to influence program comprehension were the depth of the program's hierarchy, the subject's knowledge of the program's purpose, and training. As long as the depth of the program hierarchy was two or less, comprehension was better for programs written in English. However, if the depth of program hierarchy exceeded two, comprehension was facilitated for programs written in Pascal.

If the subjects did not know the program's gist, comprehension was better for programs written in Pascal. On the other hand, if the subjects understood the program's gist, comprehension was better for programs written in English. In other words, the subjects could use the computer language to help them understand the specifics of an unfamiliar program. However, they could only use the natural language to help them understand the specifics of a familiar program.

Subjects were exposed to the programs, written in different languages, during different experimental sessions. This order interacted with the program language to influence program comprehension. Recognition memory and cloze scores of subjects who read the programs written in English first were the same or worse during the subsequent presentation. However, these scores improved when the subjects read the programs written in Pascal first.

Results of experiment two also suggested that language ability and programming experience influenced program comprehension. Program "understanding" was investigated for four classes of subjects. These four categories consisted of students who were native or non-native English speakers, who were above and below the mean in Pascal production ability, who were above and below the mean in programming experience, and who were above and below their mean in self-

rated reading ability. Use of English as a programming language for non-expert programmers was not optimal for some types of problems and some categories of programmers. In particular, applications which required more than two levels of program hierarchy were particularly difficult to comprehend if the program was written in English. In addition subjects who rated themselves as poor readers and subjects who were relatively experienced using computer languages were able to use features of Pascal to facilitate comprehension. The precise definitions, relatively simple grammar, and lack of ambiguities of a computer language compared to a natural programming language may account for their advantages in these situations.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS iv

ABSTRACT v

LIST OF TABLES xii

LIST OF FIGURES. xiv

I. INTRODUCTION 1

Purpose of the Study 1

Definition of General Terms 5

Organization of the Study 7

II. REVIEW OF RELATED RESEARCH 9

Introduction 9

Models of Textual Representation in Human Memory 12

Memory for Natural Language Text 12

Memory for Computer Programs 17

Memory for Meaning and Textual Features . . . 21

Features of the Text 22

Propositions 22

Propositional Hierarchy 24

Explicit and Implicit Propositions 25

Other Textual Features 26

Subject's Knowledge of the Topic 28

Knowledge of Schema 28

Familiarity of Topic 29

Differences between subjects 31

Comprehension of Programming languages 35
Natural Programming Languages 35
Current Computer Languages 41
III. EXPERIMENT ONE: THE INFLUENCE ON PROGRAM RECALL OF GIST, EXPLICITNESS, AND PROPOSITIONAL DENSITY . . . 45
Statement of the Problem 45
Method 48
Subjects 48
Materials 49
Procedure 50
Results 51
Discussion 54
IV. EXPERIMENT TWO: THE INFLUENCE ON PROGRAM COMPREHENSION OF LANGUAGE, TEXTUAL FEATURES AND SUBJECT'S KNOWLEDGE 56
Statement of the Problem 56
Experimental Design and Terms 59
Design 59
Terms 60
Method 61
Subjects 61
Languages 63
Facilities 63
Materials 64
Procedure 66

V. RESULTS FROM EXPERIMENT TWO 70
Dependent Variables 70
Influence of Language 73
Language and Features of The Text 79
Subject's Knowledge of the Program's Purpose 95
Program Knowledge Acquired by Training . . . 103
Influence of the Session 104
Influence of the Presentation Order . . . 105
Differences Between Subjects 109
Native Language 109
Pascal Program Production 111
Programming Experience 113
Self-Rated Reading Ability 115
Combined Differences Between Subjects . . 116
Discussion 120
VI. CONCLUSION 126
APPENDIXES 134
APPENDIX A. ADA PROGRAMS USED IN EXPERIMENT ONE . 135
APPENDIX B. PROGRAMS USED IN EXPERIMENT TWO . . . 137
APPENDIX C. PROPOSITIONAL ANALYSIS OF PROGRAMS . . 153
APPENDIX D. RECOGNITION MEMORY QUESTIONS 169
APPENDIX E. CLOZE TESTS 194
APPENDIX F. SUPPLEMENTARY DATA FOR EXPERIMENT TWO 204
APPENDIX G. PL/I PROGRAMS USED FOR EXPERIMENT TWO 231
BIBLIOGRAPHY 242

LIST OF TABLES

Table

1 Influence of Propositional Density on Program Recall 52

2 Influence of Program Explicitness on Program Recall 54

3 Summary Statistics by Language for Dependent Variables 72

4 Individual Program Means for Dependent Variables by Language 74

5 Influence of Language on Dependent Variables . . . 77

6 Features of the Text: Means 80

7 Correlations of the Means for Dependant Variables and Features of the Text 86

8 Correlations of Means for Dependent Variables and Features of the Text by Language 90

9 Influence of Program Hierarchy on Dependent Variables 93

10 Influence of Gist on Means of Dependent Variables . 96

11 Means of Gist Condition by Dependent Variables and Language 98

12 Number of Subjects with Best Recognition Memory Score per Language 100

13 Influence of Session on the Dependent Variables . 104

14 Effect of Presentation Order on Dependent Variables 106

15 Program Exposure with Cloze Tests in Same or Different Language 108

16 Scores of Native and Non-Native English Speakers by Computer Language 110

17 Influence of Programming Experience on Program Comprehension 112

18 Influence of Programming Experience on Program Comprehension 114

19 Influence of Reading Ability on Program Comprehension 115

20 Features of Text by Treatment 204

21 Individual Program Means by Depth and Dependent Variable 206

22 Influence of Gist-Condition on Recognition Memory 208

23 Number of Recognition Memory Scores in each Gist Condition 209

24 Mean Program Comprehension Scores by Session and Language 210

25 Mean Cloze Scores for Each Presentation Order . . 214

26 Influence of Native Language on Program Comprehension 215

27 Influence of Program Production Ability on Program Comprehension 219

28 Influence of Programming Experience on Program Comprehension 223

29 Influence of Reading Ability on Program Comprehension 227

LIST OF FIGURES

Figure

1 Mean Recognition Memory by Program 82

2 Mean Cloze Scores by Program 83

3 Average Seconds to Correctly Answer by Program . 84

4 Average Seconds to Read by Program 85

5 Typal Linkage Analysis Relationships 89

6 Mean Recognition Memory Scores by Depth 92

7 Influence of Depth on Recognition Memory Scores . 94

8 Mean Recognition Memory Scores in N-Gist Condition by Language 101

9 Mean Recognition Memory Score for B-Gist Condition by Language 102

10 Seconds to Correctly Answer by Subject Difference Group and Language 117

11 Seconds to Read by Subject Difference Group and Language. 119

CHAPTER I

INTRODUCTION

Purpose of the Study

How do people read and "understand" computer programs? As the number of people who interact with computers continues to increase so will the demand for individuals who can understand computer programs. Computer programming is a complex cognitive skill which encompasses a wide variety of behaviors and environments. Since a large proportion of the time spent programming involves reading programs, the ability to comprehend computer programs is an important facet of the task. Programmers often further their programming skills from reading examples of other computer programs. In addition, the ability to understand computer programs is needed to correct or modify existing programs. Designers of programming languages need to incorporate knowledge of how features of the language influence the subject in order to produce effective programming languages.

Knowledge about computer program comprehension may provide insight into the cognitive processes required for other phases of the programming task. In order to discover how people read and comprehend computer programs, it is necessary to investigate the interaction of factors relevant to this process.

There has been a great deal of interesting and productive research directed toward how people comprehend text. Do the same factors which influence the reading of natural language text also influence the reading of computer programs? Computer programs may require different reading strategies since they are a specific type of procedural text different from the type of text normally used in natural language experiments. It was the purpose of this dissertation to investigate how subjects comprehend programs. This study examined the interactions between the subjects and features of the programs that facilitate program comprehension. Factors which were considered salient for the comprehension of natural language text were explored in the context of program comprehension.

Programming languages specifically designed for the ease of human use have traditionally been the product of the designer's intuition rather than the result of empirical studies. Behavioral studies which have addressed program comprehension were most often done in terms of specific features of existing programming languages. There have also been some studies which questioned the use of a limited natural language as a programming language. Many computer scientists argued that if natural languages were used as programming languages computer programs would be easier to read and write. They felt that if the need for people to

learn special purpose programming languages were circumvented, programming would be accessible to more people (Halpern, 1967; Sammet, 1966). Opponents of this view claimed that it was not the language but the concepts that make programming difficult. They argued that even if we were able to develop an unrestricted natural language compiler it would not clarify the concepts. They also suggested that the ambiguities present in a natural language would make programs even harder to read and write (Dijkstra, 1963, 1964; Hill, 1972). The question of whether subjects can understand and remember a program written in a natural language more easily than they could the same programs written in a computer language has not been resolved. Studies, such as Biermann, Ballard, and Holler (1979) and Small and Weldon (1983), which asked whether it was appropriate to use a natural language as a programming language have yielded controversial results. There is a need for further study of the conditions for which these apparently controversial findings are true. Possibly these inconsistent results suggest that program comprehension is related to an interaction between the subject, the language, and the text.

This dissertation investigated how subjects comprehend and remember computer programs. Insights gained from this study should help answer questions such as when it is

appropriate to use a natural language as a programming language and why the natural language documentation of programs is often difficult to understand. By studying the effect on program comprehension of the interaction of language type, textual features and knowledge of the subject, this study should provide a comparative link between program comprehension and the great body of research on the comprehension of natural language text.

This study questioned how features of the text and the language affect the subject's ability to understand and remember programs. It investigated this issue by examining how variables which were salient in studies of natural language text influence program comprehension. This study asked the broad question "under what circumstances do which features of the language interact with knowledge of the subject to influence program comprehension?" Specifically it contributed to knowledge of how programs are "understood" by exploring:

1. The influence on program recall of the interaction between the subject's knowledge of the program's purpose and properties of a computer language.
2. The effect on program comprehension of the interaction between features of the text, attributes of the subject, and training using natural (English) and a typical (Pascal) programming languages.

Definition of General Terms

The following are definitions of many of the general terms referenced by the studies reviewed in chapter two:

Chunk. A meaningful unit for the interpretation of text.

Cloze test. Measure of comprehension in which every nth word of the text is deleted and subject tries to supply the word.

Lexical density. Ratio of the number of lexical words such as nouns, verbs, and adjectives to the total number of words.

Macrostructure. The highest level of understanding in the hierarchy of the text. Information is integrated between the statements to produce the overall gist.

Microstructure. Propositional representation of a passage.

Proposition. Combinations of word concepts one of which is a predicate, or a relational term, and the rest are arguments. A convenient way to think of a proposition is the least amount of textual material necessary to express an idea.

Propositional Hierarchy. The position of the proposition in the structure of the text is determined in the following two ways.

a. Repetition of arguments. A proposition which shares an argument with a superordinate or first order proposition is in the second level of

hierarchy. Propositions in common with the second level and not the first form the third level, etc.

b. Depth of hierarchy. A proposition is superordinate if it influences whether other propositions are performed. For example a loop proposition extends control over the hierarchy level of the body of the loop.

Redundancies of Text. Distributional characteristics of text predicted from other features of the text. The way that syllables contain a vowel sound is an example of redundancy.

Schema. Internal data structure used to integrate related bits of information. Also thought of as packets of information about a specific subject. Integrated pieces of knowledge organized into packets which contain knowledge about a specific subject. A schema is considered a necessary ingredient of the memory framework which guides comprehension.

Strategy The way in which the subjects use their knowledge of textual constraints or redundancies of the language to help predict subsequent text.

Subordinate. A proposition is subordinate to a higher proposition if it contains an argument that also appears in the higher proposition.

Superordinate. One or more propositions which express the most important ideas in a text base. Subordinate propositions give information about these ideas.

Surface Features of Text. Form and content of the text which provide comprehension clues. Important surface features of the text include size, density, and depth of propositional hierarchy.

Text Base. The memory representation of the meaning of the text. This consists of a partially ordered propositional structure. Superordinate propositions represent the main ideas of the text and subordinate propositions embellish these ideas.

Word Concept. Lexical entities which are combined to form propositions.

Organization of the Study

The study is organized into six sections. Chapter one provides an introduction. Chapter two reviews the current research in the areas of models of textual representation in human memory, influence of textual features on comprehension, attributes of the subject, and comprehension of computer programs. The third chapter describes the first experiment which investigated the influence on program recall of the subject's knowledge of the gist, features of the text and program explicitness. Chapter four describes the purpose and the methodology of experiment two, which

examined the influence on program comprehension of the interaction between program language, textual features, the subject's knowledge of the program's gist, training, and differences between the subjects. Chapter five reports the results of experiment two. Chapter six consists of the conclusion, implications, and suggested questions for future research.

CHAPTER II

REVIEW OF RELATED RESEARCH

Introduction

The methodologies of experimental psychology combined with theories derived from cognitive psychology, artificial intelligence, and linguistics have all contributed toward several models of how humans comprehend text. The consensus of these studies suggested that text comprehension was influenced by a combination of the features of the text and the knowledge of the subjects.

Research on text comprehension includes memory models of the representation of text. These models establish a theoretical framework for the way programs might be represented in human memory. How the text is represented in memory is important because successful recall and comprehension depend to some extent on how well the subject has been able to integrate the text with his existing knowledge of the program. Propositional memory models are an advantage for this study because they are so closely related to the way artificial programming languages are written. It is possible to write programs using a propositional construct and then convert them to either a natural or a programming language. Kintsch, 1974, posed a memory model for textual representation which consists of a

text base. Kintsch's (1974) natural language model was used by Atwood and Ramsey (1978) to suggest how subjects understand computer programs. Text comprehension research indicates that although memory for the text is primarily encoded by meaning there are residual memory traces of the features of the text itself.

Natural language studies have suggested several textual features which have been salient for the comprehension of natural language text. Included among these are variables of size (the number of words, lines, and propositions) and density (lexical and propositional). In addition, other properties of propositions such as position in the hierarchy of the text and whether they are explicitly stated also influence comprehension (Kintsch and Keenan, 1973; Kintsch, Kozminsky, Streeby, McKoon, and Keenan, 1975; Kintsch and Monk 1972; Atwood and Ramsey, 1978).

The subject's knowledge or schema of the text determines how much of what is read can be understood and recalled. Variables which contribute to the schema are the subject's familiarity with the topic, knowledge of the "theme" of the passage, training, and differences between the subjects. Differences between subjects which could influence program comprehension could be different levels of reading ability and programming experience.

Some of the considerable research that has been done in the area of human-computer interaction includes studies relating to the comprehension of computer languages. Several studies have investigated the effect of either the programming language or statements within a programming language on program understanding. The purpose of these studies is often to compare the effectiveness of constructs within existing computer languages. Much of this research has been spurred by the controversy concerning the usefulness of a natural language as a programming language. In addition, several studies on the comprehension of computer languages have had as their purpose the evaluation and comparison of languages or language features from current computer languages. Differences between subjects has been considered as a leading cause of the variance in textual comprehension for both natural and computer languages.

Discussion of the research is divided into the following topics:

1. Models of textual representation in human memory.
    a. Memory for natural language text
    b. Memory for computer programs
    c. Memory for meaning versus memory for text

2. Features of the text.
    a. Propositions
        (1) Propositional hierarchy
        (2) Implict and explicit propositions
    b. Other textual features
3. Subject's knowledge concerning the text.
    a. Knowledge of passage "theme"
    b. Familiarity of topic
    c. Differences between subjects
4. Programming languages.
    a. Natural language programming languages
    b. Current computer languages

This review is of selected studies from the extensive literature available on the comprenension of text and programming languages which are relevant to this dissertation.

Models of Textual Representation in Human Memory

Memory for Natural Language Text

How well text is remembered depends largely on how well what is read is interpreted. Recent research attributes the mechanism for this interpretation to an internal data structure or schema which is used to integrate new and related bits of information about a topic. Norman (1982) describes a schema as an integrated piece of knowledge

organized into packets which contain knowledge about a specific subject. Graessar (1980) listed the following functions of schemata: (a) providing background knowledge for context or interpretation, (b) providing knowledge for generating inferences, (c) generating expectations, (d) guiding attention and directing it toward deviations from the expectation, (e) correcting and adjusting the interpretation by perceiving and monitoring feedback. Schemata may contain both specific facts and rules for generating those facts. For example the schemata for a person reading a computer program would include: (a) the rules of the programming language, (b) the method or algorithm used, and (c) the overall purpose of the program.

A specific model of comprehension for natural language text was advanced by Kintsch (1977). This model included the memory traces for sentences and for the organization of text. He divided this organization of text into microstructure and macrostructure components. Variables that correspond to components within sentences such as letters, syllables, words, propositions and syntactic variables were defined as microstructure components. Macrostructure components which integrated information between sentences consisted of variables such as familiarity, narrativity, and new argument nouns.

Kintsch and Van Dijk (1975) describe a schema as an outline of empty slots. For example, the schemata of a story would consist of a set of expectations about the contents of the story. When the expectations are fulfilled, the outline becomes the macrostructure of the text. Macrostructure propositions are superordinate propositions in the hierarchy and describe the semantic aspects of a story.

Kintsch's (1974) model of human memory for text was based on the idea that the basic unit for textual representation in memory was the proposition rather than the word. He proposed that text was broken down into word concepts which were joined together by certain rules to form propositions. An ordered list of these propositions formed a text base. Propositions were considered to be the least amount of text that is needed to define an action or a relation. Propositions can sometimes be thought of as a subject and verb with an optional object. For example, "The fish swims" is a proposition, while "The fish" is a word concept. An example of the propositional representation and hierarchial diagram of a sentence is taken from Kintsch (1974). Given the following sentence:

A. Romulus, the legendary founder of Rome, took the women of the Sabine by force.

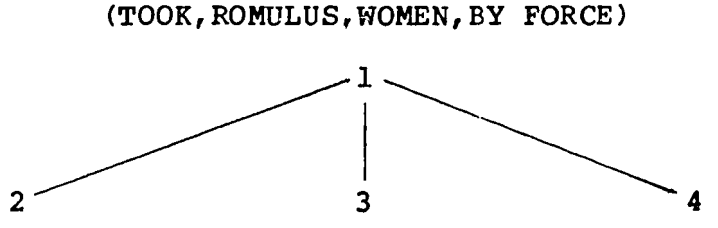
The four ideas of this sentence are:

1. Romulus took the women by force.
2. Romulus founded Rome.
3. Romulus is legendary.
4. The women are Sabine.

Kintsch's four propositional representations were:

1. (TOOK, ROMULUS, WOMEN, BY FORCE)
2. (FOUND, ROMULUS, ROME)
3. (LEGENDARY, ROMULUS)
4. (SABINE, WOMEN)

The first term in each proposition is the predicate and the rest are arguments. A diagram of the propositional hierarchy is as follows:



A second sentence, from Kintsch (1974) which has more than one level of propositional hierarchy is the following:

B. Cleopatra's downfall lay in her foolish trust in the fickle political figures of the Roman world.

Suggested propositional representations for this sentence were:

1. (BECAUSE, a, b)

2. (DOWNFALL, CLEOPATRA) = a

3. (TRUST, CLEOPATRA, FIGURES) = b

4. (FOOLISH, TRUST)

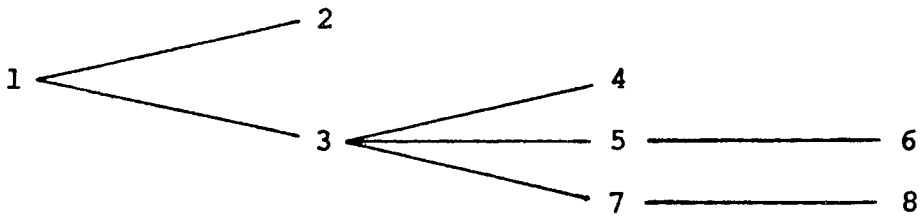
5. (FICKLE, FIGURES)

6. (POLITICAL, FIGURES)

7. ((PART OF), FIGURES, WORLD)

8. (ROMAN, WORLD)

The diagram of the propositional hierarchy is as follows:



Proposition 1 is superordinate to propositions 2 and 3 while proposition 3 is superordinate to propositions 4, 5, and 7. Proposition 5 is in turn superordinate to proposition 6 at the same level that proposition 7 is superordinate to proposition 8.

Kintsch's (1974) research shows that the higher a proposition is in the hierarchy the more likely it is to be recalled. For sentence B, subjects were most likely to remember propositions 1, 2 and 3, that Cleopatra's downfall was due to her trust in figures. The next most likely propositions to be remembered were 4, 5, and 7.

The concepts of propositionial representation and propositional hierarchy have been used to advance models of how computer programs could be represented in human memory.

Memory for Computer Programs

Recently, there has been some research concerned with comprehension of computer programs. Atwood and Ramsey (1978) demonstrated that the theories and techniques of cognitive psychology can be sucessfully applied to aspects of computer programming. They used the model developed by Kintsch (1974) to explain program comprehension and the ability to find existing errors in computer programs. They postulated that the program is represented in the programmer's memory as a partially ordered hierarchy of propositions. Their results suggest that the location of an error in the propositional hierarchy of the program significantly affects the programmer's performance in comprehension and the ability to find errors.

As an illustration of propositional analysis applied to computer programs, consider the following example of a portion of a FORTRAN program from Atwood and Ramsey (1978):

```
C     CALCULATE THE NUMBER OF CATEGORY INTERVALS
            N = (UBO(3) - UBO(1)) / UBO(2) + 0.5
C     CALCULATE TOTAL FREQUENCY
            T = 0.0
            DO 110 I = 1,N
 110        T = T + F(I)
 115        NOP = 5
 120        JUMP = 1
```

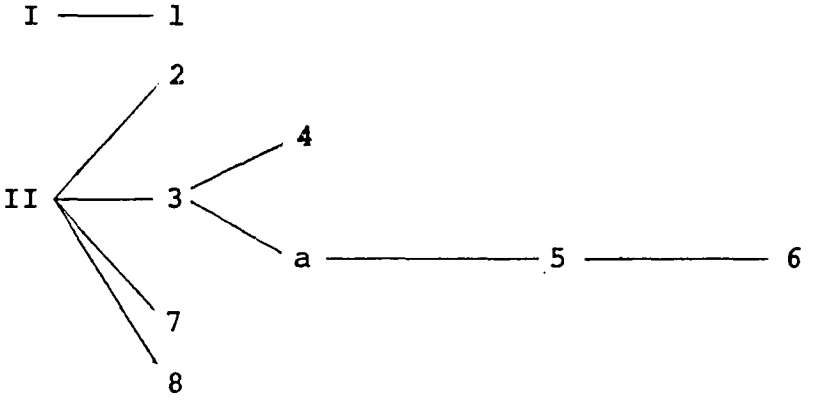
Atwood and Ramsey (1978) represented the propositional structure of this program as follows:

1. SET (N, VALUE((UBO(3) - UBO(1)) / UBO(2) + 0.5)
2. SET (T, 0.0)
3. DO (I, 1, N, a)
4. SAME (N, 1)
5. INCREMENT (T, F(I))
6. SAME (T, 2)
7. SET (NOP, 5)
8. SET (JUMP, 1)

The letter "a" in proposition 3 represents the depth of the hierarchy. It is not a specific proposition but it controlls propositions 5 and 6. The representation of the propositional hierarchy for this program segment is:

I. CALCULATE NUMBER OF CATEGORY INTERVALS

II. CALCULATE TOTAL FREQUENCY



Cooke and Bunt (1975) have postulated a model related to reading a computer program. As the subject reads the program, any memory structures or schemata which pertain to the program will be retrieved. The subject's knowledge of the program's purpose and rules of the language allow them to form an abstraction of the program and to update the relevant memory structures.

Shneiderman and Mayer (1979) have also suggested a model for program comprehension. In their model, the programmer, aided by syntactic knowledge of the programming language, constructs an internal hierarchial memory structure that represents the program. The highest level in the structure is the overall knowledge of the programs purpose such as finding a median or a square root. At lower semantic levels, the programmer may recognize sequences of statements. The greatest depth of processing would result if many levels were encoded, but one can understand the program's purpose without understanding the sequences of statements and vice versa.

Chunking, the term which refers to a concept of a stategy for encoding text into memory, is the mechanism some researchers believe that humans use to organize text into coherent groups (Miller, 1956). The grouping of elements together to form higher level units enable humans to increase their memory capacity. Norman (1982) gave the

following example of how information can be organized into different meaningful units. Remembering the sequence of individual numbers 9162536496481 is a difficult task and most people can not remember it for very long. However, if the sequence is presented as $3^2 4^2 5^2 6^2 7^2 8^2 9^2$ most people can remember it. In the first example the unit of memory is thirteen apparently random digits. When the series is restructured to the second example, the unit of memory is a sequential series of squared numbers beginning with three and increasing to nine.

Shneiderman and Mayer (1979) claim that chunking is the mechanism that enables programmers to recognize the function of groups of statements. These statements are pieced together into greater chunks until finally the program is understood. Once the internal structure of the program is organized in the subject's memory, it can be transformed into different language mediums. If this implication is correct, the language type would be a greater influence for the encoding of the text into memory than for the retrieval of text from memory.

Frost (1975) suggested that there are two bases, abstraction and function, with which subjects form a chunk. Abstraction is a generalizing process in which items are chunked based on their similiarity. A functional chunk does a specific task. Larger chunks can be based on combining

functions. Chunks based on function are important to program comprehension. The ability to remember long programs may be due to this chunking mechanism. The difference between novice and expert programmers may be in part determined by the size and contents of their chunks of memory.

Memory for Meaning and Textual Features

Although the overall knowledge of the gist of a text is a significant predictor of comprehension and memory, some studies indicate that semantics and features of the text make separate contributions to retention (Marks and Miller, 1964). Garrod and Trabasso (1973) used latency data to demonstrate that surface structure of the text as well as the meaning existed in memory.

Anderson (1974A) found evidence that both the verbatim (exact image of the text) and a propositional (meaning based representation) were remembered. Subjects studied sentences isolated and in context. They were then asked to recognize the truth of the test sentence immediately or after a delay. The verification latencies indicated that both modes of sentence representation existed. The verbatim image was strongest immediately after the presentation. When a delay was introduced, the propositional representation was dominant but there was still some memory for the exact text. This result could be particularly significant for the way in

which computer programs are remembered. Even if the meaning of the program is not immediately known, the exact text could be remembered and this might allow the subject to defer comprehension until more of the program is read.

Features of the Text

Propositions

In addition to the propositional models of memory for text, results from several studies implied that propositions are an important variable for text comprehension. Bransford and Franks (1971) performed a study to test the memory for propositions constructed from many different sentences. A short time after they presented the original sentences, they presented test sentences. These test sentences were of three types: (a) identical to the original, (b) consistent in meaning and form, and (c) consistent in form but not in meaning. Subjects had trouble identifying whether sentences had been presented before or were just consistent with the meaning of the original sentences. Subjects could correctly identify whether or not the sentences which were consistent in form but not in meaning had been presented. The sentences incorrectly identified as the ones which were originally presented were integrated propositions. The following are examples of the sentences which were presented.

A. "Three turtles rested on a floating log and a fish swam beneath them."

B. "Three turtles rested beside a floating log and a fish swam beneath them."

Each sentence was presented to a different group of subjects. The subjects were then asked whether they had seen the sentence, "The fish swam under the log." Significantly more subjects who had seen sentence A agreed than subjects who had seen sentence B. Bransford and Franks suggested that this semantic integration implied subjects established the same network structure of propositional components regardless of how component propositions were presented in sentences.

Evidence that the proposition is an underlying structure of text in memory has also been suggested by Kintsch and Monk (1972). In this study, subjects read a paragraph that required them to chain inferences. There were two types of paragraphs presented, one which stated the underlying propositions directly, and one which contained syntactic and semantic transformations of the material. The subjects required more time to read the transformed version, but there was no difference in the time required to make the inference for either type of paragraph.

Gillespie (1983) used the propositional model of Kintsch (1974) to investigate whether propositional and syntactic complexity had an effect on reading comprehension.

Although her sixth grade subjects did not have significantly different comprehension scores based on the sentence complexity, her eighth grade subjects did. Other natural language studies also indicated that subjects convert to a propositional form for later reconstruction (Anderson and Bower, 1973; Frederikson, 1972; Kintsch, 1974; Norman and Rumelhart, 1975).

Linguistic studies often paralleled the investigations of the psychologists. Moeser (1975) found that a series of words are better remembered if they describe a single conceptual unit rather than a set of conceptual units. Her study supported the results of Kintsch and Keenan (1973) which indicated that conceptual units or propositions are the basic element in the structure of text in memory.

Propositional hierarchy

Kintsch and Keenan (1973) found the amount of time for a subject to read text could be predicted from the number of propositions. Kintsch et al. (1975) discovered that the number of propositions recalled was positively related to the reading time. They also found variability in the time subjects took to read and recall the propositions. One factor which contributed to this time difference was the number of word concepts. They found longer reading times and more recall errors for texts which contained many

different word concepts. Another source of variablility in the time to read was the position of the proposition. Superordinate propositions were recalled better than subordinate ones. This finding supports the observation of Johnson (1970) that the parts of the sentence that were rated as most important for the entire text were easier to recall. Kintsch and vanDijk (1978) found that comprehension of text could be predicted with a model which included the hierarchical position of the proposition.

Explicit and implicit propositions

Subjects usually constructed a text base from explicit cues found in the text. However, they often included propositions from their schema which were not present in the surface structure of the text. Implicit propositions are supplied by the subject from the text and from memory. Explicit sentences usually took longer to read since there were more propositions. However, verification times were longer for implicit than for explicit sentences. The following examples of explicit and implicit sentences were given by Kintsch (1974).

Explicit: Police are hunting a man in hiding. The man is Bob Birch, whose wife disclosed illegal business practices in an interview on Saturday

Implicit: Police are hunting a man in hiding. The wife of Bob Birch disclosed illegal business practices in an interview on Saturday.

The explicit sentence contains information that must be inferred in the implicit sentence, namely that Bob Birch is the man who is hiding. In order to encode the implicit propositions, (HIDE, MAN) and (MAN, BOB BIRCH), it is necessary to have information such as: (a) Bob Birch has probably done something illegal (b) police look for men who have disobeyed the law (c) Bob Birch is probably in hiding and (d) Bob Birch's wife has probably disclosed his illegal business practices.

Explicit and implicit propositions can also occur in computer programs. For example, consider the following Fortran fragment from Kernighan and Plauger (1978):

```
      DO 14 I=1,N
      DO 14 J=1,N
14    V(I,J)=(I/J)*(J/I)
```

In order to understand what is being assigned to the array V, one needs to know the implicit properties of integer division contained in the Fortran language. Instead of assigning all the values of the array V to one, V would only be the value one when I and J were equal. The other elements of V would be equal to zero since Fortran automatically truncates integers.

Other textual features

Several studies have investigated the effect of specific textual features such as size and density on reading comprehension. One study examined the combined

effect of textual features on processing time. Graesser, Hoffman, and Clark (1980) collected reading times for 275 sentences. The data were scaled on nine dimensions using multiple regression techniques. The variables used included the microstructure components and macrostructure components suggested by Kintsch. As predicted, these studies showed positive correlations between reading time and the number of letters, syllables, words, propositions, new argument nouns and syntactic complexity. They also showed a negative correlation between both reading time and passage familiarity, narrativity and serial position of the sentence in the passage.

A feature of the text which warrants further discussion is textual difficulty. Cloze procedures have been used to determine the readability of natural language text. Riley (1973) has presented a selected annotated bibliography of many studies that sucessfully used this metric as a measure of reading difficulty. This method of computing text difficulty has advantages for this study because cloze tests can be easily applied to programs written in both natural and computer languages. In addition the cloze metric can also be used as to predict comprehension because it includes semantics as well as knowledge of the grammar of the language.

Britton, Westbrook, and Holdredge (1978) examined the effect of textual difficulty on reading. The difficulty of the passage was determined from scores on cloze tests. The more difficult passages took longer to read. The total cognitive capacity needed for a passage was defined as the product of the time to read the passage and the average response time to a secondary task. The total cognitive capacity was also greater for more difficult passages.

Subject's Knowledge of the Topic

Variables such as the subject's knowledge of the schema or "theme," familiarity with the topic of the text, training and reading ability all affect the subject's overall understanding of the text. Familiar, meaningful material is compatible with the existing cognitive structures and information which fits best into the conceptual framework is most likely to be remembered.

Knowledge of Schema

Several studies have indicated that knowledge of a passage's schema or "theme" is a major influence on comprehension. Dooling and Lachman (1971) suggest that the generic idea of a passage is used to facilitate the retention of prose. Bransford and Johnson (1972, 1973) conducted experiments that showed the importance of the frame of reference or schema on recall of prose text. The

following passage from Bransford and Johnson (1972) is an example of how comprehension can suffer when the schema can not be activated.

> The procedure is actually quite simple. First you arrange things into different groups depending on their makeup. Of course, one pile may be sufficient depending on how much there is to do. If you have to go somewhere else due to lack of facilities that is the next step, otherwise you are pretty well set. It is important not to overdo any particular endeavor. That is, it is better to do too few things at once than too many. In the short run this may not seem important but complications from doing too many can easily arise. A mistake can be expensive as well. The manipulation of the appropriate mechanisms should be self-explanatory, and we need not dwell on it here. At first the whole procedure will seem complicated. Soon, however, it will become just another facet of life. It is difficult to foresee any end to the necessity for this task in the immediate future, but then one never can tell. (page 722)

Once the subjects were given the passage title, "Washing Clothes", they had an appropriate frame of reference or schema for the text. Given the title of the passage, comprehension increased and subjects were able to recall twice as much information.

Familiarity of Topic

Anderson, Spiro and Anderson (1984) wrote two stories, one about eating at a restaurant and the other about shopping. Both stories mentioned the same 18 food items in the same order. However, the subjects who read the restaurant story recalled more food items than the subjects

who read the shopping story. Since the restaurant story contained more structure than the shopping story, they suggested that information was better recalled if presented in an relational framework.

Spilich, Vesonder, Chiesi and Voss (1979) asked subjects with similar verbal ability but high and low knowledge of baseball to read and recall a story about baseball. High-knowledge subjects recalled more information than low-knowledge subjects. In addition to remembering less information, the low-knowledge subjects omitted many items of tactical importance to the game.

Gagnè, Yarbrough, Bell, and Weidemann (1984) investigated the role of familiarity on recall. They found that familiarity affected the speed of the original learning and the amount of recall, independent of the learning. They hypothesized that highly familiar material has a more fully developed schema which allows more elaborate coding. Anderson and Reder (1979) also found familiarity aided comprehension. Their research suggested that any factors which stimulated elaborative processing produced more pathways in memory which subsequently facilitated retrieval of the information. Black (1983) investigated the effect of semantic and structural aspects on performance of several cloze measures. She found that both topic familiarity and sentence structure influenced reading comprehension.

Differences between subjects

Cromer (1970) advanced a difference between subject's model of reading comprehension. He found that some readers have adequate intelligence, language skills, and vocabulary skills but still have difficulty comprehending reading material. He hypothesized that the comprehension difficulty for some subjects was due to the way they organized their reading input. He suggested that instead of organizing the reading material into meaningful phrase units, some subjects used a word by word organization for their reading material. This assumption was tested by presenting subjects with reading material written using different formats. When material was presented by phrases instead of by words, the subjects defined as poor input organizers performed as well as good readers.

The ability to organize material in a meaningful way can also be a distinguishing characteristic between programmers with different levels of experience. Sheppard, Bailey, and Bailey (1984) manipulated the formats and language of computer software specification documents with the experience level of the programmers to investigate the effect on writing, finding errors, and modifying software. They presented the subjects with four types of design languages: (a) ideograms, (b) a program design language (PDL), (c) normal English, and (d) abreviated English. The

subjects were also exposed to three arrangements of spatial formats: (a) sequential; (b) hierarchical; (c) branching flow charts. The results of this study suggested that the combined use of the succinct program design language (PDL), branching spatial arrangement, and a broad experience level resulted in the best programming performance.

Studies which have investigated the difference between novice and expert programmers on various programming skills suggest that experts were not necessarily faster finding errors, nor did they do better when modification and comprehenion of programs were required. Many of studies of computer programming have noted the tremendous variation of individual performance. Brooks (1977) devised a model of cognitive processes in computer programming that offers insight into the large performance differences among subjects with the same general background and experiences. Programming behavior was determined by rules which included only a small piece of knowledge in Brooks' model. Slight differences in training or experience may produce very different rules available to the individual. Programmers with similar experience levels vary as much as a hundred to one in the time it takes to write a program. In addition, programming time for the same programmer to write programs, which were rated as equally difficult using Halstead's metric, differed by a ratio as large as six to one (Brooks,

1975). Research reported by Curtis, Sheppard, Milliman, Borst, and Love (1979) show that seventeen percent of the variance of reading comprehension for programs was attributed to differences between the subjects.

McKeithen and Reitman (1981) investigated the knowledge structures of novice, intermediate, and expert programmers in an attempt to discern internal differences underlying similar external programming behavior. They were interested in whether experts could perceive and recall more of a program because they could organize it into meaningful chunks. They found differences in the organizations of the three skill level groups. Novice programmers organize program statements according to a variety of common language associations. Intermediate programmers mixed programming and common language associations. Experts were the most similar to each other and they used an organization based on programming knowledge. Tests of such questions as whether expert subjects were more organized or whether the depth of the organization increased with skill level were not statistically significant partly due to the variability between the subjects.

Several studies have implied that the memory for computer programs of experts is significantly better than it is for novices. We have seen from the studies on text comprehension that meaningful text is more likely to be

assimilated. Expert programmers are usually more likely to find programs meaningful than are novice programmers. Chase and Simon (1974) executed an experiment in chess board pattern memorization. They found that random patterns were remembered equally by subjects regardless of their experience. However, meaningful board configurations were remembered significantly better by expert chess players. Shneiderman (1976) performed a variation of the Chase and Simon experiment using short computer segments, subjects with varying levels of expertise, and the presented program statements in either shuffled or executable order. The subjects that comprehended the meaning of the statements in executable order were able to remember them significantly better than the subjects who read the statements in the correct order but failed to understand the statement's meaning. Just as in the Chase and Simon experiment, there was no significant difference between the subject's memory for the shuffled program segments.

Adelson (1981) tested program recall of novice and expert programmers with randomly organized statements. She found that the novices correctly recalled more statements than experts. Her explanation was that experts remembered the gist and tended to generalize. She also noted that the novices tended to group the statements syntactically while experts would group them semantically. For example novices

would first recall all the IF statements while experts would recall the segment that contained the IF statement. Subject variances often overpower all other differences.

Comprehension of Programming Languages

Some of the first applied behavioral research involved the programming environment and human performance. Since then there has been a proliferation of studies involving man-machine interaction, which include studies of programming languages. Atwood, Ramsey, and Hooper (1979) have compiled a comprehensive annotated bibliography of the studies which involve human factors of computer software.

Natural Programming Languages

Knowledge of natural language is the most influential type of knowledge that humans transfer to computer languages. There have been several researchers who suggest that a natural language should be used as a programming language. Computer languages differ from natural languages as they are not used for a subject's internal dialog, the syntax is usually context free and precisely defined, and many of the symbols used do not appear in the subject's natural language lexicon. The inherent differences between the languages along with the fact that most computer languages are not designed to allow the subject to extract the meaning from many different forms of the statement,

could cause the subject to use strategies for comprehension which are dependant on the program's language type.

There is some question as to how similar computer languages should be to natural language. When a subset of a natural language is used, it may make learning easier but if only the gist is remembered, recall of the program would be more difficult. Although human-computer communication in a natural language is the goal of many language designers, Some research has suggested that procedural information is easier to follow when it is presented as a series of steps than when it is in the prose style of natural language. Kamman (1972) found flowcharts were more comprehensible than printed instructions for procedural text.

Computer scientists have been debating the usefulness of natural programming languages for more than two decades. Opponents of natural language programming argued that the nature of man-machine communication would be hindered by the lack of precision found in this type of language (Dijkstra, 1963, 1964; Montgomery, 1972; Hill, 1972). Hill claimed that even if natural languages could be used as programming languages it would be even more difficult to write programs with any certainty of what they would do.

Simmons (1986) summarized the 1984 experiment conducted by M. Jarke et al. This research compared the use of an English data base retrieval language with SQL, a simple

formal query language. Subjects had to formulate questions to solve a set of problems in English and SQL. The subjects were able to get correct answers to the SQL questions 46% of the time. When they used English, they only had a 22% success rate. It was noted that when the subjects used English incorrectly, they could only try to paraphrase the problem. This approach rarely led to correcting the problem. Since SQL was a formal structured language, the system could provide feedback about what went wrong. Since the operations which were allowed were restricted, the system could usually provide messages which helped the subject correct his errors.

Advocates of natural programming languages envisioned computers with which a greater segment of the population could communicate without the necessity of learning a new language (Halpern, 1967; Sammet, 1966, Guiliano, 1972). These proponents of natural computer languages disagreed on whether it was better to proceed from a natural language to its practical implementation, (Petrick, 1976; Hobbs, 1977; Miller, 1981; Biermann and Ballard, 1980) or to incorporate natural language features into existing programming languages (Herriot, 1977; Hsu, 1978; Reeker, 1980). The use of subsets of a natural language for programming is presently limited to a few prototypical models. Experimental studies which examine how these language

subsets affect performance are scanty and yield conflicting interpretations.

Many studies have reported evidence of comprehension difficulties encountered by subjects who used different implementations of structured programming concepts in existing or proposed programming languages (Gannon and Horing, 1975; Gannon, 1976; Embly, 1975, 1976). Small and Weldon (1983) found subjects performed data manipulations faster using the programming language SEQUEL than they did using natural language commands. Biermann, Ballard, and Holler (1979) found subjects could produce array manipulations more quickly and accurately using an English subset than when they used the programming language, PLC. Roberts (1979) conducted extensive series of tests on how people learn and use four existing interactive text editors. She found significant differences in the speed with which tasks were learned and used. The editor closest to a natural language was used more effectively by subjects regardless of their programming proficiency.

A recent study which is germaine to this discussion is that of Dyck and Mayer (1985). They examined the comprehension response times, that is the time to read and answer questions, for eight types of program statements written in English and BASIC. In addition to the language type, the statements were embedded in programs with

different macrostructure contexts, defined as the number of other statements in the program segment. Mayer (1979) had shown how each statement could be further divided into transactions. These transactions, which could be thought of as implicit propositions, were defined as an event that occured in the computer and invloved an operation on an object at a location. For example, the statement "Create a certain number in memory space A1" is broken into three transactions. Create was the operation, the number was the object, and the memory space, A1 was the location. Their experiment used two sets of subjects. The subjects who read the BASIC statements were familiar with BASIC while the subjects who read the English statements were not. The results of their experiment were as follows: (a) The group who read the English statements had significantly longer response times for all statements, (b) there was no significant interaction between the reaction times for the same statements in BASIC and English, (c) there was a significant interaction between language type and number of transactions, (d) the macrostructure or the number of other statements in the program was significantly related to the response time for both languages, and (e) the number of underlying transactions was a predictor for the response times for both languages. The results of their research led Dyck and Myer to suggest that comprehension of procedural

statements is related to underlying structural statements common to both languages.

Barnard, Hammond, Morton, Long, and Clark (1981) have investigated subject's preferred argument order. They found that subjects with no programming experience chose direct object first order for command objects. In addition, subjects who used the direct object first commands took longer, required more help and were more error prone. The best performance was obtained by not allowing a flexible command order. Barnard et al. (1981) and Black and Moran (1982) found that preferences of computer-naive subjects did not lend themselves to effective design of command languages. Black's subjects preferred general and high frequency names for commands, but general and high frequency names were more difficult to remember when subject's were given memory tasks. Carrol (1980) found that subjects could remember command names that were congruent with their function. He also found fewer performance errors if subjects used hierarchical commands in a fixed framework.

Miller (1978) has studied ways in which subjects write file manipulation procedures using a natural language. He found the style, frequently used, first stated the action then qualified it. Programming languages tend to follow a style where the principal action is embedded in a set of conditions. Miller (1981) suggested that embedded nesting

structures account for some of the difficulties encountered by trying to express computer-like solutions using a natural language. Dixon (1980) found that humans learned and followed the "action then quantification" better than the "embedded sets of conditions" style of English commands.

The consensus of psycholinguistic research suggests that humans can process any number of syntactic forms but specifying choices can influence the accuracy and the efficiency of communication. Affirmative statements and conjunction are not only easier for humans to processes using natural language (Clark and Clark, 1977) but also for specific computer programming tasks (Miller, 1974).

In many existing computer languages relations between conditions are expressed either by nesting or branching. A number of behaviorally oriented experiments have been performed that investigate these different types of control flow. Sime, Green, and Guest (1977) found it was easier for novices to write programs using a nesting style particularly if it included markers to specify the range of a conditional. They also found an advantage for redundant specification such as IF-NOT-ELSE.

Computer Languages

The current interest of computer scientists in structured programming has influenced the type of

experiments that have been performed with specific computer languages. Shneiderman (1976) investigated such topics as comparison of logical and arithmetic IF statements, the effect of program modularity on comprehension, and the utility of comments and meaningful variable names. Although Shneiderman's results were not statistically significant, Weissman (1974) found comprehension was aided by program form. Factors that were believed to make programs complex such as flow of program control, flow of data, and program form were tested by modification tasks and comprehension tests. Results from Weissman's study conflicted with those of Shneiderman. Weissman's study implied that program form (factors such as indenting and comments) aided program comprehension.

One of the most ambitious experiments in contrasting designs of programming languages was done by Gannon and Horning (1975). They made nine modifications on a computer language which they had written for teaching. These modifications were based on what was considered to be desirable features for structured programming. The overall error rates between each of the compared languages were not significantly different but some types of errors were found to be significant by post hoc analysis. These errors were operator precedence, the assignment operator, use of semicolon as a separator, use of brackets to close compound

statements and expressions, and difficulty using named constants. This experiment did not question the psychological basis for these results.

Much of the research on "understanding" text involves either the percent of correct responses or the response time. Often these studies appear to yield conflicting results. If measures are made on only one dependent variable, the results may be task specific. However if the criterion variables include response time as well as percent correct, it may be possible to gain a broader picture of the processes involved in the comprehension of text.

In the process of establishing a methodology for studying computer programs Weissman (1974) found first semester programming students encountered problems with the constructs of the programming language but second year students were able to extract the program's meaning. This review suggests a need to systematically investigate the effect on program comprehension of the interaction of features of the text, the subject's knowledge of the program's purpose, training, and differences between subjects. This dissertation will explore which factors best predict the subject's ability to comprehend programs by examining the variables which have been shown to influence how people read and comprehend natural language text. The broad objective of this study is to investigate the effect

on program comprehension of the various properties of the text and attributes of the subject. In an attempt to ascertain how humans read programs, subjects currently taking a second course in programming languages were studied for this dissertation. Although the variance of the subject's programming experience is less for this study than it usually is in computer language related research, differences between subjects are still expected to influence program comprehension without sublimating any other effects. Any insights into program comprehension which may be derived from research using second semester programming students may be extended to other experience levels in the future.

There have been some studies which investigated the features of existing computer languages. Generally, they tried to find a superior construct within that language for subjects with varying backgrounds. The present research differed from previous studies in that:

1. It decreased the task dependency of the experiments by sampling program "understanding" with several measures which are correlated with the processing of natural language text.
2. It used subjects with similar backgrounds to investigate how program comprehension was influenced by knowledge of the gist, training, language ability, and programming experience.

CHAPTER III

EXPERIMENT ONE: THE INFLUENCE ON PROGRAM RECALL OF GIST, EXPLICITNESS, AND PROPOSITIONAL DENSITY

Statement of the Problem

Variables which were found to be salient in the research on natural language text comprehension included the subject's knowledge of the text's topic, the amount of explicit information provided by the text, and surface features of the text. Surface features, which have been identified as significant in the these studies, were the properties of the text such as size (the number of words and propositions), lexical density (the ratio of nouns, adjectives, and verbs to the total number of words), and propositional density (the ratio of the number of propositions to the number of words). Since propositions or the "essence of a sentence" were used as variables in experiments conducted for both natural language text and computer programs, they were selected as an independent variable for experiment one. The propositional density (PD) or the ratio of the number of propositions to the number of words or symbols was used in order to allow programs to be different sizes.

Programs were written so that they conformed to low and high propositional density, and had "output" which was explicit or implicit in the program's text. The program's

"output" differs from the algorithm. The algorithm, which can be extracted from the program text, is considered to be "how the program achieves the result." The "output" of the program cannot always be extracted from the algorithm. For example, the program's "output" may be a list of numbers, sorted in ascending order. There are several algorithms that could achieve this result. However, each algorithm would consist of a different series of program statements. An explicit program is one in which the algorithm could be used to derive the program's "output". Programs which are explicit have program outcomes which are transparent to the subject. An example of an explicit program is one in which a subject is able to tell from the statements of the program that the program's purpose was to find the largest element in the array. Implicit programs are those in which the knowledge of the program's "output" are opaque to the subject. The "output" of implicit programs must be deduced from the subject's knowledge about the program rather than from the program's text. An example of an implicit program is the computation of a Fibonacci series. The process of adding the next number in sequence to the accumulated sum can be found in the algorithm, but external information must be provided from the subject's knowledge of the topic in order to relate the algorithm to the computation of a Fibonacci series.

The dependent variable was a binary variable which categorized subjects by whether or not they were able to recall 95% or more of the program. Program explicitness or transparency, propositional density (PD), and the knowledge of the program's "output" (GIST) were the independent variables.

Experiment one was designed to ascertain if program recall of second semester programming students was influenced more by knowledge of the program's "output" or whether features of the text could facilitate recall. This experiment specifically focused on the relationship of the subject's knowledge of the program's "output" (GIST), the propositional density of the text (PD), the amount of textual explicitness or transparency of the program's "output" and its influence on the subject's ability to recall the program within one programming language. Specifically this study asked: (a) Does propositional density (PD) interact with the subject's knowledge of the program's "output" (GIST) to influence recall? and (b) Does the explicitness or transparency of the program's "output" interact with the subject's knowledge of the program's "output" (GIST)?

It was hypothesized that recall would interact with propositional density and the amount of explicitness in the program. The subject's knowledge of the program's purpose

should predict its recall. That is, when the subjects knew the program's purpose, they had a well developed schema or cognitive framework with which to organize the text for memory. If subjects did not possess this overall knowledge, features of the text such as propositional density could be important facilitators for program recall. Just as beginning readers are sometimes able to comprehend individual sentences but not understand the meaning of a larger unit of text such as a paragraph or passage, second semester programming students may understand separate program statements but not comprehend the program's "output."

Method

Subjects

A section of thirty-two students who were taking the University of Hawaii's second computer science course agreed to participate as subjects for experiment one. Second semester subjects were chosen because they were able to read programs for meaning. All subjects had previously completed one computer course in which they had written between ten and twenty programs using the programming language, BASIC. The median age of the subjects was twenty-two. Twenty of the subjects were male and twelve were female. English was the native language for all of the subjects.

Materials

Four programs were written that satisfied each of the explicitness and propositional density requirements. The transparency of the programs "output" or whether it is implicit or explicit depends on the subject's knowledge about that program. In order to find an appropriate set of explicit or implicit programs for experiment one, the subjects were asked to read and describe the program's "output" for several programs. The programs chosen as explicit had at least 95% of the subjects familiar with the program's output. Those which were chosen as implicit had fewer than 5% of the subjects familiar with the programs "output."

The other manipulated feature of the program was the propositional density, (PD). Six explicit and six implicit programs were converted to propositional form using a methodolgy similar to the one used by Atwood and Ramsey, (1978). The mean propositional density for these twelve programs was 25%. Programs with the lowest and highest PD from each of the explicit and implicit categories were selected for the study. Although low PD programs were were defined as those with a PD of less than 25% and high PD programs were defined as ones with a PD greater than or equal to 25%, the mean of the low PD programs used was 16.5% and the mean of the high PD programs used was 33.5%.

The high PD programs had one level of nesting, that is a loop within a conditional statement or a conditional statement within a loop. Programs which contain nesting are defined to be hierarchical. Atwood and Ramsey (1978) have suggested that the position of propositions in the hierarchy influences program comprehension. Thus, the hierarchy was balanced in the propositional density (PD) factor.

The programs were written using the programming language ADA. Although none of the sujects knew ADA, they had learned a similar language, PL/I, during the semester. A similar language was used in order to enable the subjects to understand the meaning of the statement manipulations. Since they had not learned the ADA language, they would have to remember the text rather than use the language's rules to generate the text.

The programs were from six to twelve lines long. The number of tokens (variable names and symbols) in each line varied. During the first part of the semester the algorithms and output from each of the programs had been used as examples.

Procedure

All thirty-two subjects participated in the experiment simultaneously. They were each given a booklet which consisted of four ADA programs surrounded by blank pages.

These programs can be found in Appendix A. The subjects were told to try to remember the program exactly as it was written. They were also told that they could not refer back to the programs once they had turned the page. The subjects were given the following instructions:

> Read the following programs one at a time. Spend as long as necessary trying to memorize exactly what is written. When you feel you are ready, try to reproduce the program segment on the next blank page. Once you turn the page do not refer to the program again.
>
> After you are through writing what you can remember from each program, turn to the next page and write the overall purpose of the program or its output. Don't specify the algorithm just what the program is supposed to accomplish.

There was no time limit for each segment but a total of fifty minutes was given. All the subjects were able to complete the task in that time.

Results

The dependent variable was the subject's ability to accurately recall 95% of the program. Tallies were also obtained on whether or not the subject understood each programs "output" (GIST). Appendix A shows the programs and the relationship between the subject's knowledge of the program's "output" and the subject's recall. The probabilities of the 2 by 2 contingency tables were calculated exactly using a computer program available from the University of Hawaii's Department of Educational

Psychology program library. Statistically significant differences of $p < 0.001$ existed between the number of subjects who could either recall or not recall the high propositional density programs.

These results suggest that program recall was facilitated if the subjects knew "what the program does." Only the subjects that knew the program's output were able to reproduce it to criteria. Subjects who lacked the knowledge of the program's output could still reproduce the low propositional density programs. Table 1 shows the number of subjects in each propositional density, gist, by recall group.

Table 1

Influence of Propositional Density on Program Recall

		Propositional Density			
		Low		High	
		gist	no gist	gist	no gist
95% Recall	yes	29	29	31	9
	no	2	4	3	21

Analysis of the effect of the program's transparency suggested that program recall was influenced by an interaction between the subject's knowledge of the program's "output" and the explicitess of the program "output." Results, shown in Table 2, suggest that when the subjects knew the program's output, most of the programs were recalled but more explicit than implicit programs were remembered. Possibly this is because the algorithms of the explicit programs describe the program's "output." This knowledge of the program's "output", in turn, enhanced the probability of successful recall.

Examination of the recall data when the program's purpose is not known indicates that implicit programs are recalled 2.2 times more frequently than explicit programs. Anderson (1974) suggested that implicit sentences require the subject to provide some type of schema. If the subject is not aware of the program's correct purpose, the program needs to be carefully read to provide a framework for the program recall. Implicit programs may be more successfully recalled than explicit programs because a greater depth of processing is required to form alternate schemata.

Table 2

Influence of Program Explicitness on Program Recall

		Gist			
		Yes		No	
		Explicit	Implicit	Explicit	Implicit
95% Recall	yes	35	25	12	26
	no	4	1	13	12

Discussion

The high propositional density (PD) results are consistent with those of McKeithan et al. (1981); Mayer(1981); Adelson (1983) and Shneiderman (1976). A high level schema was needed to allow the subjects to chunk the text and reduce their memory load. If the subjects knew the overall purpose of the program, they possessed a better knowledge stucture to help organize the material and thus, they were able to recall more of the exact program text. However, the low PD programs show that the knowledge of the program's "output" is not always essential. Low PD programs can be recalled without a highly developed schema to help organize the text in the high PD programs. Without a highly developed schema, features of the text such as propositional

density and program explicitness may facilitate program recall. The influence of propositions on the recall of natural language text has been suggested by several studies (Kintsch, 1979; Kintsch and Vipond, 1977; Kintsch and van Dijk, 1978; Vipond, 1980; Atwood and Ramsey, 1978). Royer, Lynch, Hambleton, and Bulgareli (1984) showed that propositional density correlated significantly with subject's ability to recognize the statement's meaning.

The suggestion that implicit semantic cues could facilitate recall is supported by natural language research. Anderson (1974) has suggested that sentences that require inference on the part of the subject require more thought and that this effort has the effect of making the programs easier to recall.

This first experiment suggested that features of program text such as propositional density and explicitness of semantic cues can influence program recall when the program's overall purpose is not known. Features within a language can aid memory. Will this be true regardless of the language of the program? The next phase of this study examined the influence on program comprehension of language, textual features, and the subject's knowledge.

CHAPTER IV

EXPERIMENT TWO: THE INFLUENCE ON PROGRAM COMPREHENSION OF LANGUAGE, TEXTUAL FEATURES AND SUBJECT'S KNOWLEDGE

Statement of the Problem

How can the programming language or components of the language facilitate an understanding of the program? Many computer scientists involved in the design of programming languages espouse an eventual goal of simplified man-machine communication. The view that this would be achieved by using a natural programming language is based more on intuition than on empirical evidence. While the same program written both in a computer and a natural language can have similar semantic content, the programs may contain several textual differences which could differently affect the comprehension of the program.

Studies from the fields of psychology and linguistics suggest variables which influence the comprehension of natural language text. Included among these variables are several which are text-related such size (the number of words, the number of lines, and the number of propositions) and density (lexical and propositional). Other influences include variables which differentiate subjects. Some of these subject-related variables are their knowledge of a program's purpose, their training, their language ability, and their programming experience. The emphasis of this

experiment was to determine which text-related and subject-related variables would predict program comprehension and memory. Experiment two addressed the question: Under what conditions do properties of a language interact with subject-related variables to influence comprehension and memory for programs? The propositional density (PD) conditions in experiment one were balanced with respect to the depth of nesting. These variables have been separated in experiment two.

To test the hypothesis of interaction between language and textual features, the same programs were written using both natural and computer languages. Unrestricted English was used rather than an existing natural language subset to provide a comparison of the results with implications suggested from the reading research.

There has been very little research directed toward understanding the factors that influence reading computer programs. Experiment two was designed to provide evidence which might suggest the kinds of interaction that exist between the subject, language, and the features of the text on program comprehension. Factors such as program language, program form, program size, program difficulty, familiarity of the subjects with the program, training, and subject's ability and experience could all influence the comprehension and memory for computer programs.

Guided by experiment one and the literature review this study investigated:

1. How the programming language interacted with textual features and subject-related variables to influence program comprehension and memory?
2. Which textual differences between computer and natural languages best predicted comprehension and memory for programs?

Specifically this study asked:

1. How do textual features such as lines, tokens, propositions, propositional density and textual density influence program comprehension?
2. Does the depth of the propositional hierarchy have an influence on whether a computer or a natural language better facilitates program comprehension?
3. Does the subject's overall knowledge of the specific purpose of the program and the programming language interact to influence program comprehension and memory?
4. Does training affect the comprehension of computer programs?
5. Can differences between the subjects such as native language, programming experience, program production ability, and reading ability predict which language facilitates program comprehension?

It was hypothesized that there would be an interaction of language, features of the text, and abilities of the subject which would influence the ability to understand a program. Subjects are expected to have better comprehension and memory for programs written in Pascal when: (a) the programs contain several levels of embedded nesting, (b) the overall purpose of the program is unknown, (c) the subjects are poor readers, and (d) English is not the subject's native language. Conversely subjects are expected to understand programs written in English better when: (a) the programs have at most two levels of embedding; (b) the overall purpose of the program is known; and (c) the subjects are not proficient in Pascal.

Experimental Design and Terms

Design

The experimental design used repeated measures on two languages and twelve programs. Each subject had two reading sessions with the order of the language and the problem counterbalanced across subjects. The subjects read twelve programs in each session. The first session used six programs written in Pascal and six different programs written in English. The second session was held at least twenty-four hours later and used the same programs written in the opposite language.

Terms

The following are the definitions of the terms relevant for this experiment:

Comprehension. The thinking process which enables meaning to be extracted from the text. Measures of this process were obtained from: (a) The percent of answers correct on an experimenter designed instrument administered to the subject after each treatment (RMEM), (b) A question given after each treatment to determine if the subject knew the purpose of each program (GIST), and (c) A cloze test in which an underline character was substituted for every fifth word or symbol of every program (CLOZE). The score on this test was determined by the percentage of correct symbols, words, or synonyms of the word.

Time spent on storage and retrieval of text. This time was divided into: (a) Storage time (SREAD) - the number of seconds that a subject took to read each treatment, and (b) Retrieval time (SAC) - the average number of seconds that a subject took to correctly answer the comprehension questions for each treatment.

Readability index (RIDX) - the average cloze scores on each treatment obtained from a population of approximately two hundred second semester programmining students prior to experiment two. This readability index was used to balance the program difficulty between sessions.

Textual Density. The average number of tokens per line. This is also a measure of the space surrounding the text.

Training. (SESSION) Experimentally manipulated prior exposure to the treatments.

Surface features of the text. Physical attributes of the text. Measures of the textual characteristics of each treatment include: (a) The number of lines (NLIN), (b) The number of words or symbols (NTOK), (c) The textual density which was defined as the number of words or symbols per line (DEN), (d) The number of propositions (NPROP), (e) The propositional density (PD) which was defined as the ratio of the number of propositions to the number of symbols in the program, (f) The depth of the hierarchy (HIER) in each program (the levels of nesting which were held constant for programs written in both languages).

Method

Subjects

Seventeen male and six female subjects were randomly selected for this study from the University of Hawaii's second computer course, ICS 267, the semester following experiment one. This course differed slightly for the two experiments. The subjects learned the language Pascal instead of PL/I but the emphasis of the course, during both

semesters, was on the development of algorithms. During their first computer programming course, the subjects had all used the programming language, BASIC, to write approximately ten programs. This study was designed to investigate subjects with similar backgrounds. However, subjects were asked to provide information about variables such as their age, sex, grade, prior computer experience, perceived reading ability, and native language which might differentiate them. In addition, each subject was given a quiz on Pascal program production as part of their class work. Two small programs were graded with a maximum of three points each. The average grade for the subjects used in experiment two was 3.0, with most of the subjects receiving partial credit for each problem. The average number of years of college which the subjects had was 2.5 or halfway between the sophomore and junior years. The mean number of programs written, including those written during the current semester, was 30. The mean age of the subjects was 21. Although all of the subjects spoke English fluently, three males and two females spoke Chinese as their native language. Ten subjects rated themselves as good readers, five subjects considered themselves to be very good or expert readers, and eight subjects rated themselves as fair or poor readers.

Languages

The languages used for this study were the natural language of English and the computer language of Pascal. The subjects all learned Pascal from the experimenter in the same second semester computer science course.

Facilities

Experiment two was performed in the microcomputer laboratory of the University of Hawaii's Department of Information and Computer Sciences. Two sound-proof rooms contained the three microcomputers which were used. Each microcomputer contained a programmable clock and either two eight inch floppy disk drives or one hard disk and one eight inch disk drive to record the data. The microcomputers were placed, next to Zenith model 19 green screen terminals, on desk tops that extended across the room. Padded desk chairs in front of each terminal were adjusted to a comfortable height for each subject. All the software, developed for this study by the experimenter, used a PL/I compiler written by W. Wesley Peterson (1981). Appendix G contains a listing of the Pl/I programs used to present the stimulus material.

Separate files were made on the disk for each subject at each experimental session in order to record every response typed. The experimenter provided a session and program set code. Based on this code, the software picked the appropriate set of programs and randomly assigned either

six English programs followed by six different Pascal programs or six Pascal programs followed by six different English programs.

Materials

Twelve algorithms were written in both Pascal and English. They were divided into two sets of six English and six Pascal programs. All the programs in each set were unique, but the two sets were complementary: that is, set one and set two had the same algorithms but used different languages. The experimental design balanced the order of the language and the program set. Program difficulty was balanced between sets by the readability index. The set number and the language initially presented were assigned by the experimenter during session one. Then during session two, the subjects read the programs in the opposite language and language order. Within the six programs of one language, the order of the programs was randomly assigned by the controlling program.

The validation of the programs and the consistency between the two languages was provided by first converting the twelve algorithms to propositional form using methods described by Kintsch (1974) and Gillespie (1983). The programs were then written in Pascal and English from the propositional forms. See Appendix B and C for the listings

of the programs and the propositional analyses. Ten ICS faculty and second year graduate students acted as judges for the programs. They either read the English and wrote the appropriate Pascal or read the Pascal and wrote the English equivalent. Since the propositional form was very similar to the Pascal version, the judges were able to quickly agree on the Pascal programs. The English versions took much longer but after several changes, 95% of a different second year programming class produced correct Pascal programs from the English version.

A recognition memory test was written by the investigator. Several multiple choice questions were made for each of the twenty-four programs. One of the questions for each program specifically asked the program's overall purpose. Five ICS graduate students read the questions and suggested changes and clarifications. A pencil and paper version of the programs and the edited version of the test was given to twenty students in different sections of the second level programming course as a pilot study. An item analysis of the questions eliminated the ones which contributed the least variability. The final test contained five multiple choice questions for each program. The recognition memory questions were designed to be similar for the same program in the two languages. Appendix D contains a listing of the comprehension questions.

After the last question of the last program at each session, the subject was thanked for participating. The experimenter then gave the subject a pencil and paper cloze test. The cloze test, shown in Appendix E, was used to test the extent that the language facilitated the subject's schema. Subjects were randomly assigned to either a cloze test in a same or an opposite language condition. For the same language condition, the subjects took a cloze test on the programs they had just read. For the opposite condition, the cloze test used the programs of the opposite language. The order of which language provided a subject's first exposure to the program was recorded. Synonyms were counted as correct answers.

Each program was given a readability coefficient which was based on the average percent of correct responses from the cloze scores of two hundred second year programming students who took the cloze tests prior to experiment two. The readability index was used as a way to balance the program difficulty across sessions and as measure of external validity.

Procedure

The subjects were tested individually during a week at the conclusion of of their second computer course. The experimenter had a desk in an office outside of the micro-computer rooms. The experimenter assigned subjects to

vacant terminals when they arrived at their scheduled times. The experimenter keyed in a code which determined the initial language and the program set number. The controlling program requested that the subject enter a session number (1 or 2) and his or her last name.

The experimenter explained to the subject that the purpose of this research was to study how people read and understood programs. The subject was advised that the programs would be presented one at a time and that there was no time limit for reading the programs. The subject was told not to take notes but to try to understand the program and remember it exactly as there would be questions on both the content and form of the program. If the session number was 1, a sample program appeared on the sceen. The subject was instructed:

> Spend as long as you wish reading this program,
> press return when you are finished.

Once the subject pressed return, a sample multiple choice question appeared on the screen. The subject selected an answer. The experimenter asked if the subject nad any questions about the procedure. The experimenter then left the room and the controlling program presented each program followed by questions on the screen. The programs were randomly presented to the subjects. That is they were either given English programs first or Pascal programs first

but within each language the order was random. After each program, this message appeared at the bottom of the screen:

"PRESS RETURN WHEN YOU THINK YOU CAN REMEMBER THIS PROGRAM"

The program clock was started once the program appeared on the screen and it was stopped when they pressed return. As soon as the subject pressed return, the program dissapeared and a multiple choice question concerning the program appeared. As soon as the subject answered the question, both the answer and the time taken to answer were recorded on the subject's file. After five multiple choice questions, the next program appeared on the screen. This procedure was completed for six English and six Pascal programs during session one. Data for each subject were kept for later analysis.

There were two similar sessions for each subject. The sessions were at least twenty four hours apart. All subjects read all twenty-four programs but they never had the same program in both languages during the same day. At the end of each session, the subject was given a cloze test. The subjects were randomly assigned to two cloze conditions, same or different, The same condition subjects took a cloze test on the material they had just read, while the opposite condition subjects took a test on the program in the opposite language. Both groups of subjects were instructed:

> There are several programs on both sides of the following page. Please fill in what you feel is the correct symbol where a "_" appears. One dashed line is given for each missing symbol or token. This information will help determine program difficulty.

The subjects were also asked to fill in their age, sex, grade, perceived reading ability, and experience with Pascal and programming. Whatever program set and cloze test the subject was given in session one were recorded by the controlling program, and the complementary set was given in session two.

CHAPTER V

RESULTS OF EXPERIMENT TWO

Dependent Variables

Experiment two was designed to investigate the effect of four types of variables on the subject's ability to "understand" programs. These predictor variables were: (a) textual features which are inherent in natural and computer languages, (b) subject's knowledge of the program's purpose, (c) prior training, and (d) characteristics of the subjects. The experiment was designed to explore the interaction of these predictor variables with the language type.

This experiment uses many of the dependent variables which have been successful in natural language research. Criterion variables used to assess program "understanding" were: (a) scores on a recognition memory test (RMEM), included in this measurement were binary measures of whether the subjects understood the overall purpose of the program (GIST), (b) average number of seconds taken to correctly answer questions from the recognition memory test (SAC), (c) the number of seconds that the subjects took to read each treatment (SREAD), and (d) scores on a cloze test (CLOZE).

Some of these variables predominately measured the subject's overall knowledge or cognitive framework. The recognition memory questions (RMEM), for example, included a question concerning the overall purpose of the program

(GIST). The average number of seconds the subjects took to recall the correct answers (SAC), was a measure of the time needed to retrieve the information from memory. It was anticipated that the time to correctly answer the recall questions (SAC) would also reflect components of the subject's overall knowledge of the program. It was hypothesized that longer retrieval times might suggest a more complex internal representation. RMEM and SAC also relied on the subject's memory for the treatment. SREAD was a measure of the time the subject took to read the text, to remember it, and to internally store it for future recognition questions. The cloze score was the variable least dependent on the subject's memory. Instead it measured the subject's ability to predict missing "tokens" from knowledge derived from a combination of content and syntactic rules of the remaining text. In a sense the cloze measure reflects the overall knowledge as well as contributions of the language to program "understanding."

Table 3 shows the means and standard deviations by language for each of the dependent variables. The CLOZE, RMEM, and GIST scores are percent correct (out of 100) while the SAC and SREAD are the number of seconds.

Table 3

Summary Statistics by Language for Dependent Variables

	Means				
	CLOZE	RMEM	GIST	SAC	SREAD
English	76.79	60.00	50	22.16	120.77
Pascal	82.58	56.74	46	22.51	163.96

	Standard Deviations				
	CLOZE	RMEM	GIST	SAC	SREAD
English	21.03	27.40	50	18.35	101.24
Pascal	17.03	25.49	50	20.02	111.93

N (23 Subjects reading 12 Programs each languages)					
	CLOZE	RMEM	GIST	SAC	SREAD
English	258	276	276	276	274
Pascal	270	276	276	276	275

In general, subjects had better recognition memory and understood more of the program's purpose for programs written in English. However, they received higher cloze scores for programs which were written in Pascal. The subjects spent more time reading and answering Pascal programs. Once the programs were read, the mean number of

seconds needed to correctly answer recognition memory questions for the programs was almost the same for both languages. These results suggest that subjects may have language independent internal storage structures which consist of an abstract, representation similar to the representation they have for natural language text.

The recognition memory scores without the gist questions were highly correlated (0.68) with the gist scores for both languages. Therefore, the gist score was included as part of the recognition memory score for most of the analyses. The time taken to store the information (SREAD) and the time to retrieve it (SAC) were moderately correlated (0.37) and thus, were analyzed as separate dependent variables.

Influence of Language

The question of whether the textual features, inherent in the languages, help the subjects comprehend the programs was examined to determine the influence of the language and textual features on program "understanding." Table 4 shows the means, by language, for each program.

Table 4

Individual Program Means for Dependent Variables by Language

Program	Dependent Variable CLOZE	RMEM	GIST	SAC	SREAD
1. Greatest Common Denominator					
English	71.95	40.87	17	15.20	109.45
Pascal	79.72	50.43	35	28.99	213.39
2. Random Number Generator					
English	73.41	50.43	22	18.87	108.62
Pascal	76.03	35.65	9	27.51	130.77
3. Factorial					
English	81.80	48.70	52	23.56	115.74
Pascal	88.07	59.13	57	28.07	104.95
4. Fibonnacci Numbers					
English	87.83	71.30	35	22.37	109.53
Pascal	90.48	73.04	30	22.28	102.41
5. Directions					
English	83.56	66.96	74	25.79	99.43
Pascal	91.59	53.04	57	18.93	95.10
6. Exponential Equation					
English	78.64	71.30	74	22.02	175.77
Pascal	81.41	73.04	87	15.17	126.63

	Dependent Variable				
	CLOZE	RMEM	GIST	SAC	SREAD
Program					
7. Square Root					
English	55.63	48.70	17	25.01	132.86
Pascal	88.78	48.70	17	17.96	197.61
8. Mean					
English	77.27	71.30	65	20.36	109.59
Pascal	72.72	63.48	57	20.67	228.26
9. Maximum Value					
English	74.48	59.13	57	24.83	134.77
Pascal	76.99	60.87	43	23.99	171.87
10. Median					
English	70.24	80.00	87	23.85	124.58
Pascal	83.52	54.78	74	22.91	149.63
11. Frequency					
English	87.63	64.35	52	21.24	69.90
Pascal	81.82	54.78	48	23.48	191.27
12. Prime number					
English	76.32	46.96	52	22.78	158.04
Pascal	80.44	53.91	35	31.27	254.02

A review of varied scores on the 12 programs also reveals that there was an interaction between the program and the language in which they were written. Analysis of the differences between the means of the dependent variables, by language and program, showed that the greatest differences were in the time spent reading the treatments (SREAD) and in the cloze scores (CLOZE). The language type (English or Pascal) appears to have an effect, but it's not clear that one language is necessarily superior for all treatments.

The next question of interest was the extent to which the language type interacted with the program. Since repeated measures were used in the experimental design, an analysis of variance was performed with the appropriate error terms. Scores were recorded for each subject on twenty-four programs, twelve of which were in English and twelve of which were in Pascal. The dependent variables used in the analysis of variance analysis were the number of seconds the subjects took to read each treatment (SREAD), scores from cloze tests (CLOZE), and recognition memory and gist questions (RMEM). Table 5 shows the results of this analysis for the selected dependent variables.

Table 5

Influence of Language on Dependent Variables

Seconds Taken to Read Treatments (SREAD)

Source	DF	Anova SS	F	PR > F
Subject	22	116.47		
Language	1	21.63	28.07	0.0001
Subject * Language	22	16.95		
Program	11	42.78	7.79	0.0001
Subject * Program	242	120.74		
Language * Program	11	34.24	3.81	0.0001
Subject * Language * Program	242	195.19		

Cloze Scores (CLOZE)

Source	DF	Anova SS	F	PR > F
Subject	22	155.21		
Language	1	11.89	5.12	0.0338
Subject * Language	22	51.08		
Program	11	38.73	6.11	0.0001
Subject * Program	242	138.93		
Language * Program	11	32.71	6.61	0.0001
Subject * Language * Program	242	98.44		

Recognition Memory (RMEM)

Source	DF	Anova SS	F	PR > F
Subject	22	124.28		
Language	1	2.11	3.29	0.0834
Subject * Language	22	14.09		
Program	11	74.22	9.97	0.0001
Subject * Program	242	163.83		
Language * Program	11	21.98	3.21	0.0005
Subject * Language * Program	242	150.50		

The variables for the time taken to read (SREAD), the cloze scores (CLOZE) and the recognition memory score (RMEM) all show a significant interaction between program and language. The results of the analysis of variance on each of the measures of program comprehension suggest that the language and the program's algorithm both contribute to program comprehension and memory. The statistically significant interactions indicate that it is not simply a case of one language being superior to another. Conditions which are included within the program's text appear to influence which language best facilitates "understanding" of the program.

Language and Features of the Text

The next phase of the experiment was to investigate properties of the programs which could explain the interaction of language and textual features. Textual features which were indicated as contributors to text comprehension in natural language research were investigated for both the English and Pascal versions of the algorithm.

The conventional use of English and Pascal dictates a high correlation among the textual features. In comparison to programs written in English, programs written in Pascal usually have more lines and fewer tokens: therefore, they have a lower textual density. Since fewer tokens are used per line, lines in Pascal tend to be surrounded by space. A computer language usually contains only the tokens which are essential for the meaning while omitting the embellishments that provide redundancy in a natural language. Four of the programs, included in this experiment, were contrary to the usual case. Two programs were written to have the greatest number of tokens in the Pascal version of the algorithm. The rest of the programs had more tokens in the English version than in the Pascal counterpart. Two different programs were constructed to contain the same number of propositions for each language. Thus, the Pascal programs all contained either the same or more propositions. All the Pascal programs had a greater propositional density. This

ratio varied from being almost the same to being three times greater for Pascal programs. The hierarchy level, however, was kept constant for the two versions of each program. The overall means of the program's textual features and the means by language are shown in Table 6.

Table 6

Features of the Text: Means

	No. Lines (NLIN)	No. Tokens (NTOK)	Textual Density (DEN)	No. Prop. (NPROP)	Prop. Dens. (PD)	Hier-archy (HIER)	Read. Index (RIDX)
Pascal	19	75	4	19	25	2.4	75
English	8	106	13	14	13	2.4	75
Combined	13.6	90.3	8.4	16	19	2.4	75

Table 20, in Appendix F, shows the surface properties of each of the treatments by language. The mean readability index was the same for both languages but not consistent within the treatments. Seven of the English programs and five of the Pascal programs had the greatest readability indexes. These indexes were derived from the percent of correct answers on a cloze test taken by two-hundred different second semester computer science students. A higher percentage of correct scores corresponds to an easier text. Figures 1 through 4 show each of the dependent

variables in order of descending English performance. The interactions are illustrated by the deviations of the Pascal measurements on both sides of the line. One can see from Figure 1 that the recognition memory scores were for the most part better for the English than the Pascal programs, but that the two programs, numbers 4 and 6, which had the same number of propositions were almost the same for both languages. Figure 2 shows that the cloze scores were generally higher for the programs written in Pascal, but the two programs, numbers 8 and 11, which had fewer tokens for the English than the Pascal version had better English cloze scores. Figure 3 shows that the number of seconds taken to answer did not appear to be influenced much by the type of language. Figure 4 shows that the number of seconds spent reading the programs was greater for the programs written in Pascal.

The programs are referenced by number (see Table 4 for the names and properties of the program text by language.)

Figure 1. Mean Recognition Memory (RMEM) by program (Means are ordered on English).

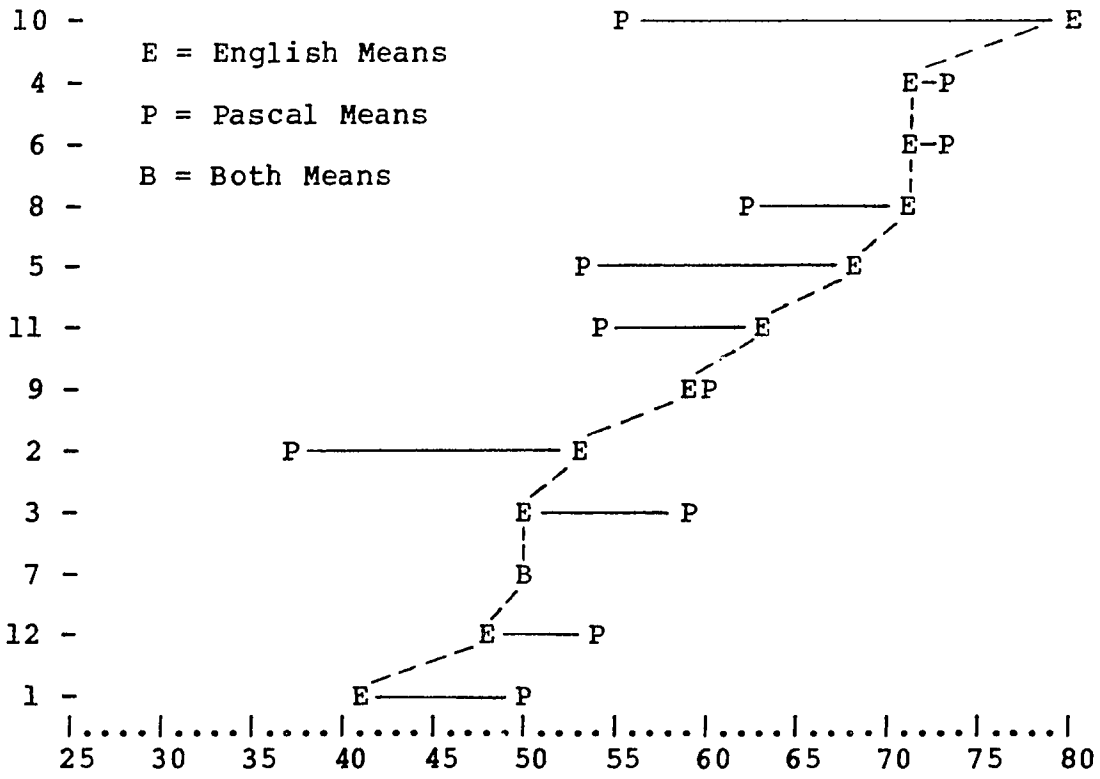


Figure 2. Mean Cloze Scores (CLOZE) by Program (Means are ordered on English).

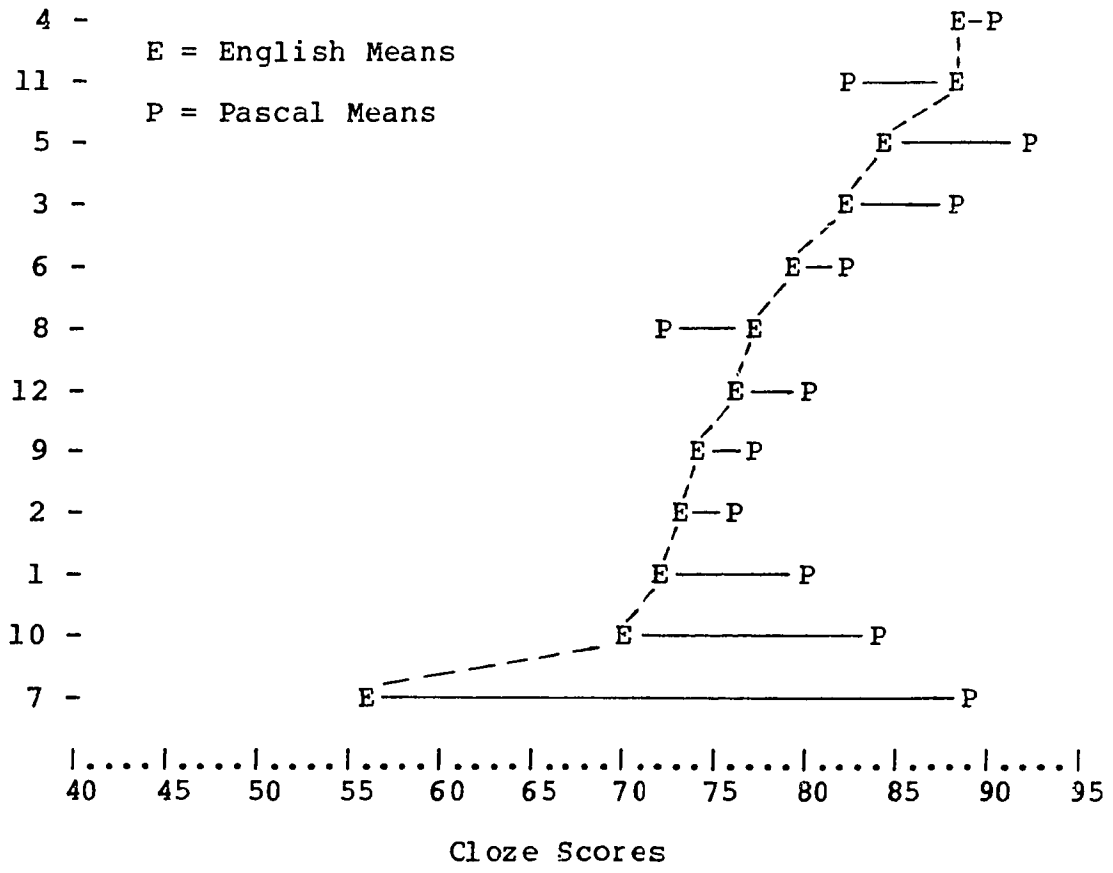


Figure 3. Mean Seconds to Correctly Answer (SAC) by Program (Means ordered on English).

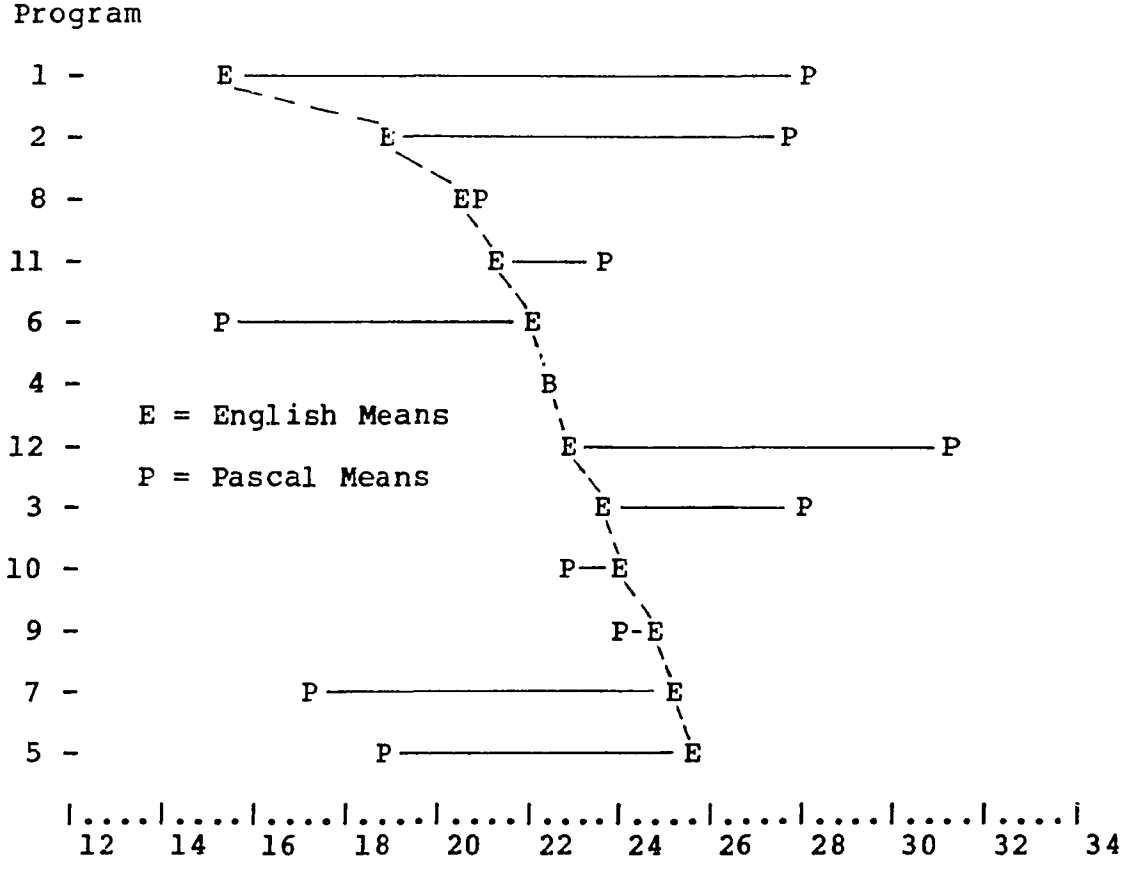
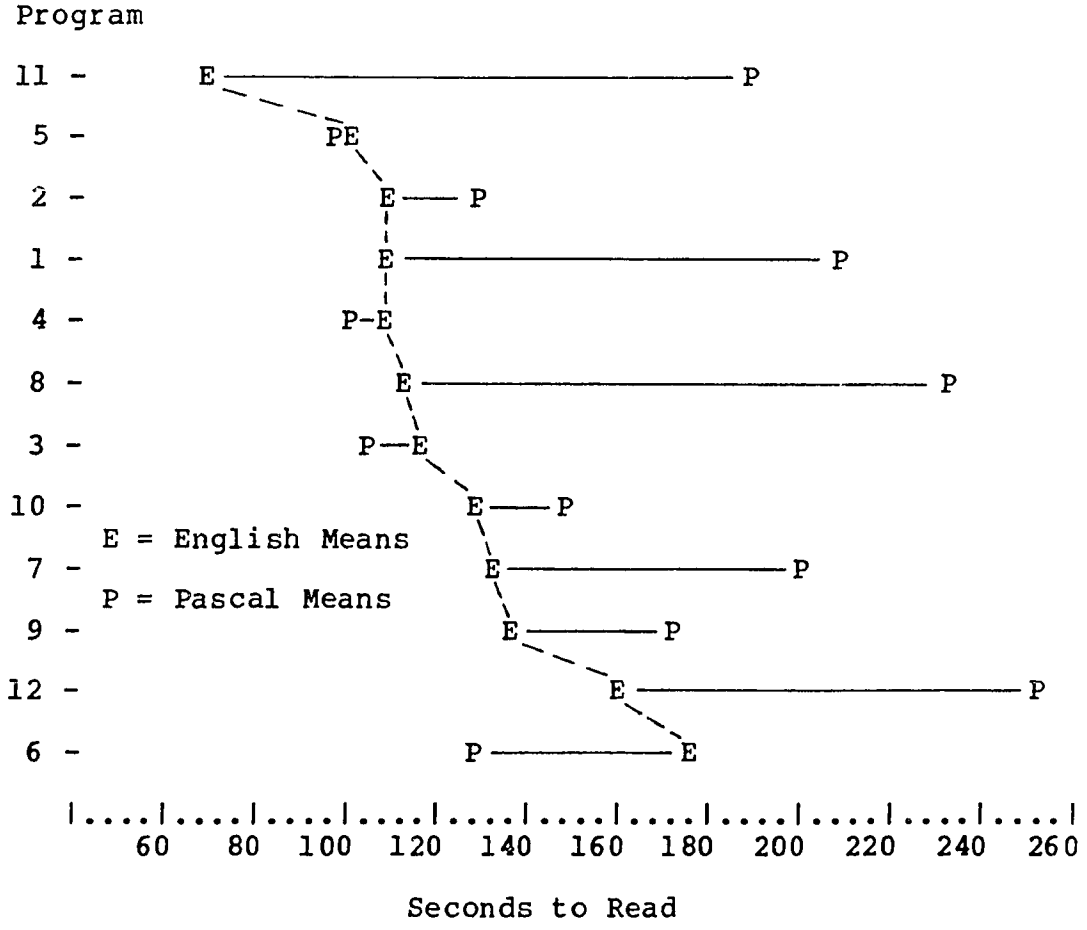


Figure 4. Mean Seconds to Read Programs by Program (SREAD) (Means ordered on English).



Correlation coefficients, shown in Table 7, were computed to determine the relationships between the dependent variables and features of the text.

Table 7

Correlations of Means for Dependent Variables and Features of the Text*

*correlations computed from means of 23 subjects reading 24 programs. Correlations >= |40| significant p <= 0.05.

	Rmem	Gist	Cloze	Sread	Sac	Ridx	Hier	Nlin	Ntok	Den	Nprop
Gist	68										
Cloze	20	15									
Sread	-18	-12	-16								
Sac	-16	-7	-18	37							
Ridx	11	10	78	-36	2						
Hier	-28	-4	-34	46	33	-24					
Nlin	-27	-11	14	72	23	-15	29				
Ntok	-15	15	-61	19	21	-35	49	-8			
Den	14	15	-44	-40	-3	-5	3	-84	56		
Nprop	-29	4	6	54	38	-3	39	69	39	-35	
Pd	-15	-9	54	32	18	28	-6	73	-46	-82	61

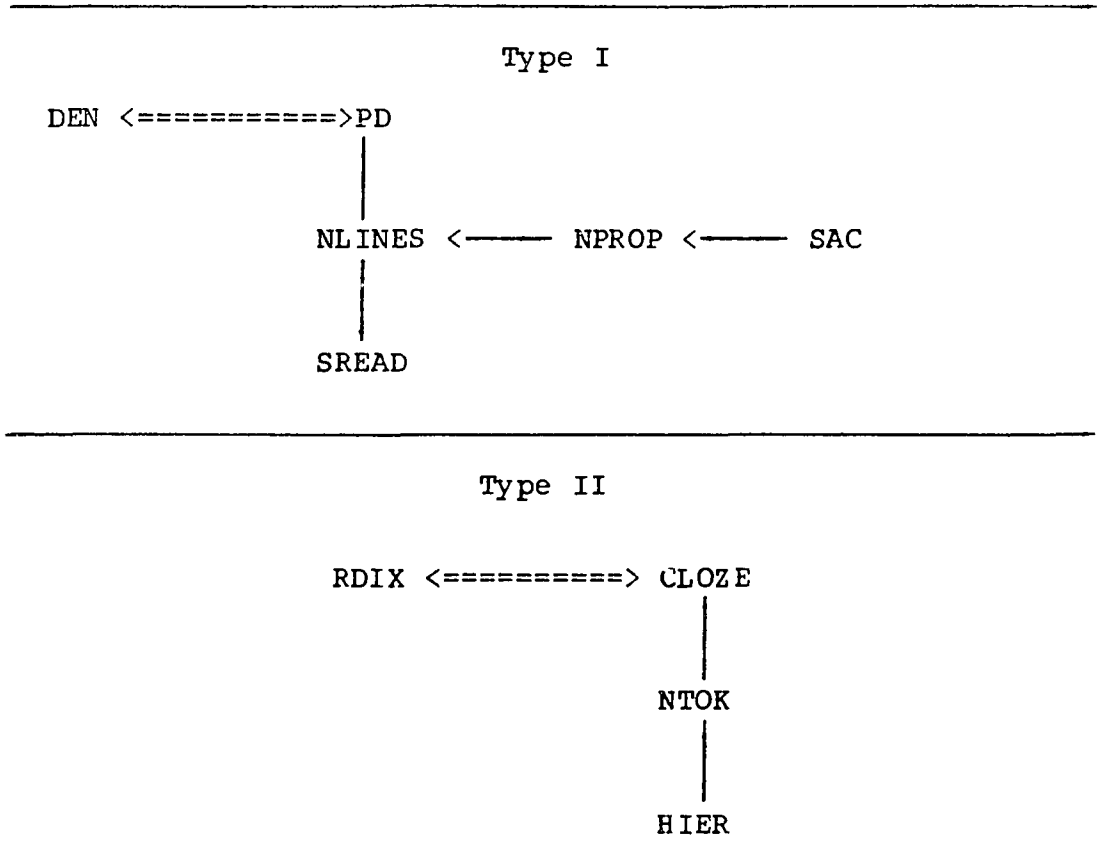
The hierarchy or nesting level (HIER) was held constant for the programs in both languages in order to test its interaction with language for each of the dependent variables. HIER increased with program size variables such as the number of tokens (NTOK), the number of propositions (NPROP), and the number of lines (NLIN). The correlations of hierarchy (HIER) showed the greatest combined effect on the the dependent variables, SREAD, SAC, CLOZE, and RMEM. Since longer and more complex programs generally take longer to read and retrieve , SREAD and SAC increased with HIER. The cloze and recognition memory (RMEM) scores were greater for programs with a hierarchy level (HIER) of one or two. The correlations suggest that HIER is an important variable in program comprehension and, as such, should be more closely examined.

The measurements of the average time taken by the subjects to read the treatments (SREAD) were correlated with variables of length and density. SREAD measures were positively related to the number of lines (NLIN) and the number of propositions (NPROP) and were inversely related to the lexical density (DEN). Interestingly, it was the number of lines (NLIN), not the number of tokens (NTOK) which exerted the greater influence on the reading time. As anticipated, SREAD times were positively but only moderately correlated with retrieval times (SAC).

In addition to SREAD, the cloze scores (CLOZE) also showed considerable association with features of the text. The cloze scores were most closely related to the readability index (RIDX), but they were also influenced by program length and density. Cloze scores were inversely related to the number of tokens (NTOK) and the lexical density of the lines (DEN), but positively related to the propositional density (PD). Since Pascal programs had a greater propositional density, this association predicts higher cloze scores for the Pascal programs. The recognition memory questions (RMEM), which was highly associated with GIST, was the dependent variable least associated with the features of the text.

Although the correlation matrix shows many strong dependencies between the variables, McQuitty's (1957) elementary linkage analysis was used to isolate the dominant factors. Figure 5 shows the three clusters which emerged. They consisted of: (a) the textual features of density, size and reading and retrieval time, (b) the readability index and cloze scores which were related to the number of tokens and the amount of hierarcial nesting, and (c) the recognition memory score and the subject's knowledge of the gist.

Figure 5. Typal Linkage Analysis Relationships.



The next question addressed the influence on program comprehension of the interaction of language type and properties of the text. Table 8 shows the correlations of the dependent variables and the program's textual features by specific language.

Table 8

Correlations* of Dependent variable Means and Features of Text by Language

*correlations computed from mean scores of 23 subjects and 12 programs, correlations >= |.56| significant p <= 0.05

English

	Rmem	Gist	Cloze	Sread	Sac	Ridx	Hier	Nlin	Ntok	Den	Nprop
Gist	74										
Cloze	36	34									
Sread	-4	19	-37								
Sac	35	48	-4	21							
Ridx	37	41	91	-43	2						
Hier	-47	-3	-22	36	-14	-11					
Nlin	-32	11	-46	38	23	-41	56				
Ntok	-31	13	-52	51	20	-47	56	97			
Den	2	18	-30	62	-4	-28	9	4	29		
Nprop	-25	15	0	18	11	-17	39	78	72	-11	
Pd	15	3	65	-38	-7	34	-21	-19	-30	-49	43

Pascal

Gist	60										
Cloze	17	2									
Sread	-20	-22	-57								
Sac	-55	-42	-41	45							
Ridx	-23	-25	77	-44	3						
Hier	-6	-4	-60	65	64	-39					
Nlin	-30	-14	-49	84	44	-26	66				
Ntok	-19	9	-51	69	38	-31	60	84			
Den	-8	31	-38	34	17	-27	31	43	85		
Nprop	-28	4	-23	54	52	6	46	77	82	63	
Pd	-23	-5	28	-1	36	57	-5	21	4	-11	59

When the analysis was separated by language, most of the correlations of textual features become even stronger. The level of nesting (HIER) was still related to the length, that is the number of tokens, lines, and propositions for both languages. However, for the English programs, the level of nesting (HIER) was inversely related to the recognition memory scores (RMEM), but for the Pascal programs, the HIER hierarchy level was correlated with the time taken to read the programs (SREAD), the time to answer the RMEM questions (SAC), and the cloze scores.

Since the correlations suggested that the hierarchy influenced the RMEM, CLOZE, and SREAD scores, the hierarchy (HIER) was examined in more detail. The four hierarchical levels used for this analysis were further separated into two categories. A hierarchy of one or two levels was defined as low depth and a hierarchy of three or four was defined as high depth. Figure 6 shows the interaction of the depth for the mean recognition memory scores. Notice how the scores are higher, when the depth is low, for the programs written in English. On the other hand, when the depth is high, the programs written in Pascal received the higher scores.

Figure 6. Mean Recognition Memory scores by Depth.

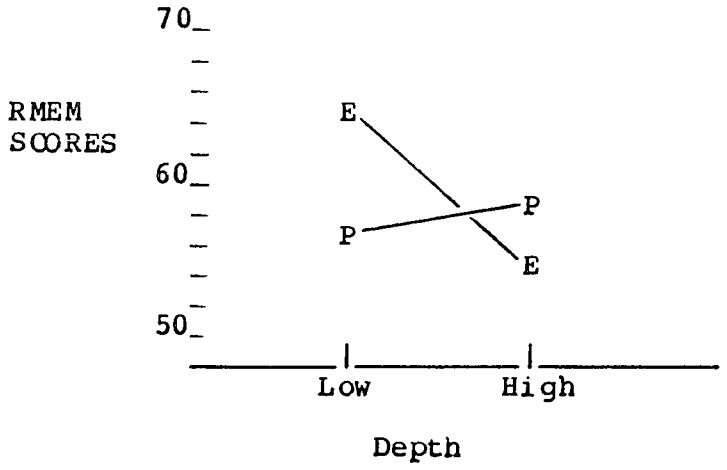


Table 21, In Appendix F, shows the individual program means by depth for the dependent variables. Table 9 shows the means of the RMEM, CLOZE, and SREAD measures by language and depth condition. The mean scores for the CLOZE tests decrease for programs with high depth. However, more time is also spent reading programs which have high depth.

Table 9

Influence of Program Hierarchy on Dependent Variables

Recognition Memory Scored (RMEM)

Depth	English	Pascal	Mean
Low	64.72	56.25	60.4
High	53.39	57.66	55.5
Mean	60.0	56.7	

CLOZE Score

Depth	English	Pascal	Mean
Low	76.71	84.80	81.0
High	76.36	79.59	78.0
Mean	76.8	82.6	

Seconds to Read (SREAD)

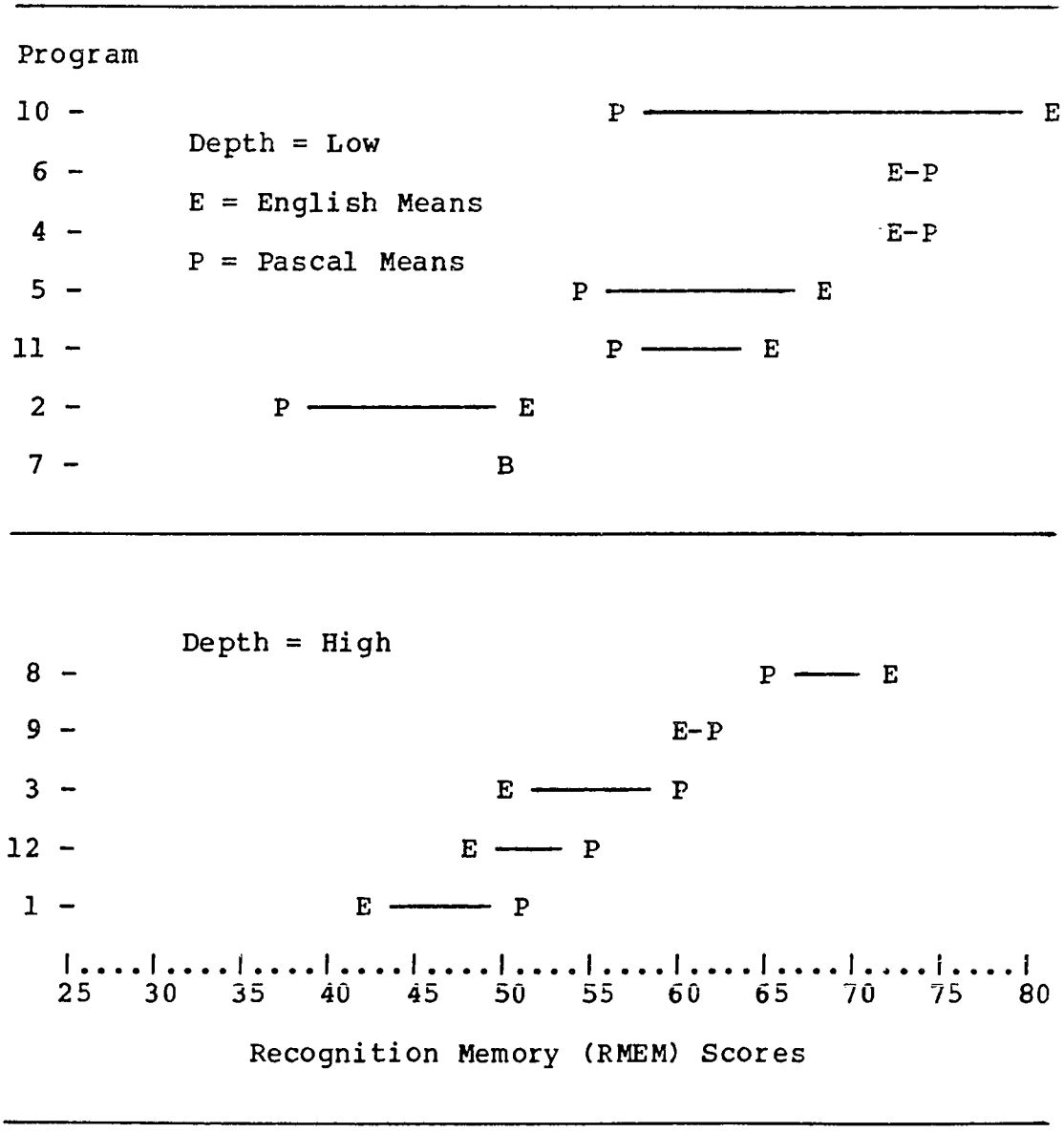
Depth	English	Pascal	Mean
Low	117.24	141.92	129.58
High	125.52	194.50	160.45
Mean	120.8	164.0	

Note. Low Depth = Hierarchy Level < = 2

High Depth = Hierarchy Level > 2

Figure 7 shows a plot of the average recognition memory score by treatment, language and depth. The programs are ordered by the English version score.

Figure 7. Influence of Depth on Recognition Memory.



For most of the programs, the English recognition memory score was higher for programs with a low depth, but when the program depth was high Pascal programs received a higher score. Exceptions for the low depth group were programs four and six, which had the same number of propositions in both the English and Pascal version. The high depth exception was program eight, which had fewer tokens in the English version than in the Pascal version. Programs seven and nine also received either the same or slightly higher Pascal RMEM scores. Program seven, with a low depth, had almost the same number of propositions in the English and Pascal versions and received the same RMEM score for both languages. Program nine, with a high depth, had 25% more propositions for the Pascal than for the English programs. As expected, the subjects did better for the Pascal version of program nine. The results suggest that program depth interacted with the language type to influence recognition memory scores (RMEM).

Subject's Knowledge of the Program's Purpose

The next question addressed was whether the ability to "understand" programs was affected by the subject's knowledge of the programs gist or overall purpose. It was shown in Table 7 that the recognition memory scores which do not include the gist have a linear correlation coefficient

of 0.68 with the gist score. Table 10 shows the means of the dependent variables by GIST.

Table 10

Influence of Gist on Means of Dependent Variables

Gist	RMEM	CLOZE	SAC	SREAD
Purpose Unknown	43.07	78.58	20.26	136.58
Knew Purpose	74.94	81.00	24.58	148.74

As anticipated, the recognition memory scores were significantly higher if the subject knew the gist. These findings indicate that, analogous to the results from natural language studies, knowledge of the program's overall purpose does significantly affect how much is remembered. If the subject was familiar with the program's purpose, the recognition memory scores (RMEM) were almost double what it was when the purpose was unknown. Subjects familiar with the purpose took more time to correctly answer (SAC) the recognition memory questions than they did when the purpose was unknown. The longer time needed to correctly answer the recognition memory questions could suggest that the subjects who knew the program's purpose had a more fully developed framework or cognitive structure that needed to be scanned before an answer was found. The subjects who knew the

program's purpose spent more time reading the the programs than subjects who did not know the purpose. Possibly more time was required to assemble information into the more complex cognitive structure or framework of subjects who knew the program's purpose. The cloze measure showed that subjects who knew the program's purpose scored higher. The group that did not understand the program's purpose performed well anyway. The knowledge of the program's purpose was not as great a predictor of success for the cloze tests as it was for the recognition memory measures. The success of the subjects on the cloze tests could be due to the nature of the measurement. The subject does not need to rely entirely on memory, since all but every fifth token in the text was available for retrieval cues.

In order to more closely analyze the effect of the knowledge of the program's purpose and any interactions with the program language, the recognition memory scores were examined by language type and the following four categories of gist: a. N-gist, the subjects did not know the gist for either language, b. E-gist, the subjects knew the gist for the English program only, c. P-gist, the subjects knew the gist for the Pascal programs only, and d. B-gist, the subjects knew the gist for both programs. See Appendix F, Table 22 and 23 for the mean scores for each gist condition by language, and the number of subjects per category

respectively. Table 11 shows the means for each of the dependent variables for each of the gist conditions.

Table 11

Means of Gist Condition by Dependent Variable and Language

		Gist Condition			
		N-Gist	E-Gist	P-Gist	B-Gist
Variable	Language				
RMEM	English	41.60	72.40	44.32	80.22
	Pascal	43.40	44.40	70.81	72.80
CLOZE	English	76.05	74.07	73.82	80.33
	Pascal	79.09	85.66	81.75	85.10
SAC	English	19.61	20.76	22.63	25.60
	Pascal	19.78	20.75	33.20	22.11
SREAD	English	105.22	116.16	123.21	140.25
	Pascal	152.50	177.67	176.63	163.74

In order to remove the subjects who knew the gist in at least one of the language presentations, the scores from the extreme gist conditions, the N-Gist and the B-Gist classifications, were investigated. The absence of overall knowledge of the program's purpose in either language

suggests that the subjects used other strategies to help them remember the programs. On the other hand, if they knew the gist in both languages, the subjects were likely to have a well developed schematic framework to aid their comprehension. The means shown in Table 11 indicate that the RMEM scores in the extreme gist conditions also interact with language. That is, if the program's purpose was not known in either language, the subjects had slightly higher RMEM scores for the programs written in Pascal. However, when the knowledge of the program's purpose was known for both languages, the RMEM scores were greater for the English programs.

Although knowledge of the program's purpose showed the greatest difference in the RMEM scores, it is interesting to note other interactions with the program language in the extreme gist conditions. The subjects took more time to answer the RMEM questions when they understood the program's purpose than when they did not. Language was not an influence in retrieval time when the program's purpose was unknown (19.6 versus 19.8 seconds). However when it was known, the subjects spent more time correctly answering the English questions (25.6 seconds for the English versus 22.11 for Pascal questions).

When the program's purpose was unknown, the subjects spent 152.5 seconds reading the Pascal programs and only

105.2 seconds reading the English programs. When the subjects knew the program's purpose in both languages, they averaged 163.7 to read the Pascal programs and 140.5 seconds to read the English programs.

The cloze scores were generally higher when the program's purpose was known. However, there was almost no difference between the Pascal scores in the N-Gist condition and the English scores in the B-gist condition. The Pascal programs have higher average cloze scores than the English counterparts.

The best recognition memory scores were tallied by language and extreme gist condition. Table 12 shows the number of subjects who received the RMEM scores for each language and each gist condition.

Table 12

Number of Subjects with Higher Recognition Memory Scores

	Purpose Unknown	Knew Purpose
English	3	8
Pascal	9	3

Figure 8 shows a plot of the average RMEM scores in the order of descending proficiency for the English programs, under the N-Gist condition. It graphically shows that Pascal programs have an edge for higher recognition memory (RMEM) scores when the program's purpose is not known.

Figure 8. Mean Recognition Memory Scores (RMEM) for N-Gist Condition by Language (Ordered on English)

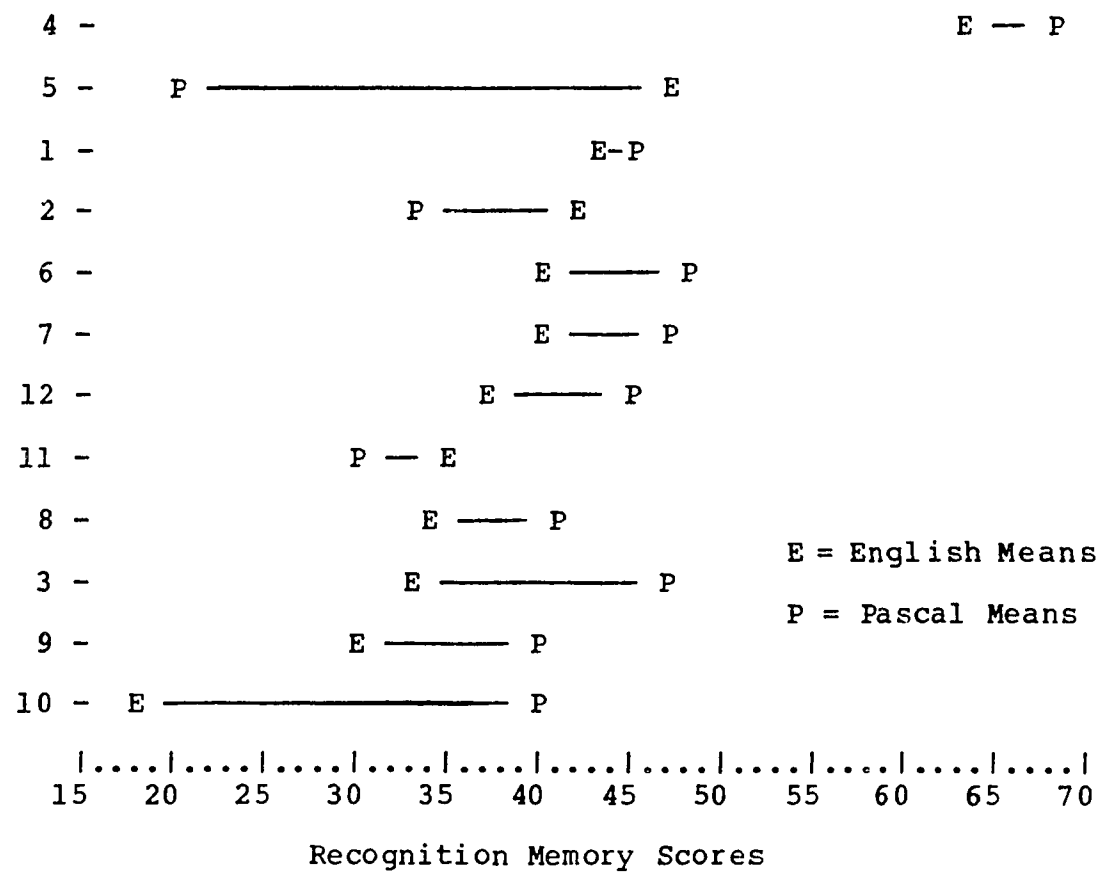
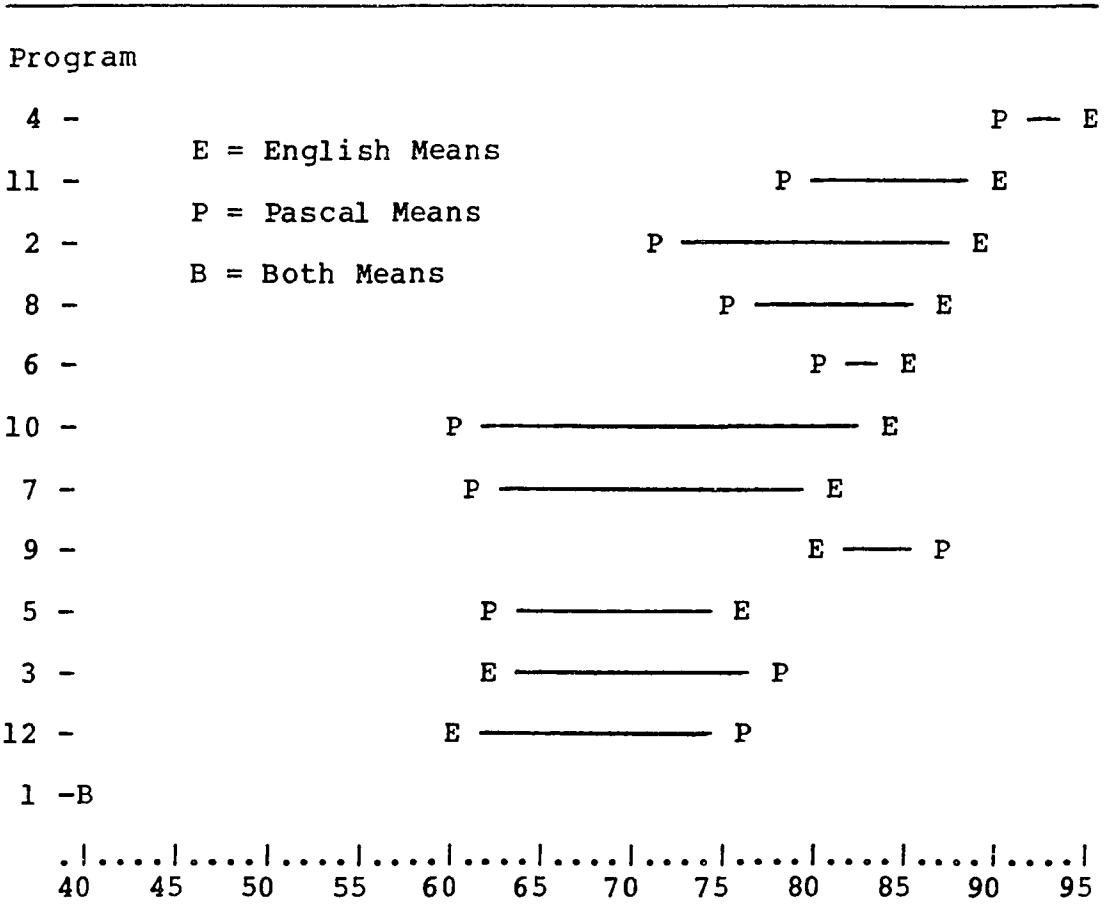


Figure 9 is a plot of the average RMEM scores when the subjects knew the gist of the programs in both languages. When the subjects knew the gist well, the RMEM scores were generally greater for the programs written in English.

Figure 9. Mean Recognition Memory Scores for B-Gist Condition by Language (Ordered on English)



Results from this section imply that if the subjects knew the program's purpose in both of the languages, they received the highest recognition memory score on the programs written in English. If they did not know the purpose for either program, they received better scores for the programs written in Pascal. Without any overall schema, it appears as though the subjects need to rely on other strategies to help them remember the program. Without the framework of the gist, a computer language, with its predictable syntax, seems more amenable to other strategies than a natural language.

Program Knowledge Acquired by Training

The next question addressed was whether training and language type would interact to influence program comprehension. It was assumed that reading the program in either language would provide prior knowledge which would be available to the subject during the next session. However it was not known whether the language type might influence the quality of the training for subsequent exposures of the programs. The type of training was experimentally manipulated by the session number and by the order of presentation of the algorithm in each language. In addition there was a within session experimental condition. The within session training consisted of two versions of the cloze test. That is, the programs in the cloze tests which

the subjects took at the end of each session were either in the same or the opposite language from their prior exposure.

Influence of the session

Six English and six Pascal programs were presented during Session I. During Session II, which was at least 24 hours later, the programs previously presented in English were presented in Pascal and the programs previously presented in Pascal were presented in English. The sessions were counter-balanced across programs and subjects. The purpose of the session treatment was to determine if prior exposure independently provided a cognitive framework that facilitated program comprehension. Table 24, in Appendix F, shows the means by program, session, and language for each of the dependent variables. Table 13 shows the means for the dependent variables by session.

Table 13

Influence of Session on the Dependent Variables

Session	CLOZE	RMEM	SAC	SREAD
I	77.17	58.33	27.80	174.87
II	82.23	58.41	16.86	110.29

Recognition memory (RMEM) scores were not affected by the session number. However, the average reading and retrieval times were lower and the average cloze score was higher during session II.

Influence of the presentation order

A further question concerned whether language type interacted with the presentation order to influence comprehension. Since the programs were balanced across sessions, the session number provides insight into a general training effect, but it doesn't specifically answer whether the presentation order of the language might influence program comprehension. To examine this question, the means were tallied by order of presentation for each language. For example, an English treatment could be presented first or it could be presented after the same Pascal program, while the Pascal treatment could be presented first or after the same English program. Table 14 shows the means by language and presentation order.

Table 14

Effect of Presentation Order on Dependent Variables

	English First	Pascal Second	Pascal First	English Second
RMEM	58.70	55.51	57.97	61.30
CLOZE	73.94	84.82	80.25	79.52
SAC	28.33	17.74	27.27	15.98
SREAD	148.98	127.51	200.67	93.07

The recognition memory scores (RMEM) were higher for the English programs after the subject had prior exposure to the Pascal programs. Prior exposure to the English programs did not facilitate the Pascal recognition memory scores. In fact the first presentation recognition memory scores were 2.5 points higher than the recognition memory scores after they were exposed to the English programs.

More time was needed to correctly answer questions for the first presentation of English programs than for the first presentation of Pascal programs. However, during the second presentation, the questions from the English programs, which had the Pascal training, were answered more quickly than those from the Pascal programs. These results support the previous results which suggested that programs written in Pascal provide a deeper cognitive framework for

the subject's later retrieval than programs written in English.

The cloze scores were improved between sessions. Reading the English version after the Pascal version brought the English cloze scores to the same level of the Pascal the first time it was read. For this variable reading the English version did seem to help the Pascal version during the next session.

The session and presentation order treatments required the subjects to remember the programs for at least 24 hours. The final experimental condition involved two types of presentations within the session. Since the cloze test was given at the end of each session, it was possible to randomly assign the subjects into two groups. One group was given a cloze test on the same material that they had just completed and the second group was given a cloze test on the treatments in a different language. Table 25, in Appendix F, lists the cloze scores by presentation order and language. Table 15 shows the Cloze scores categorized by whether or not the subject read the same program in the same language during the same session.

Table 15

Session I Program Exposure with Cloze Tests in the Same or Different Language

		Initial Exposure	
		English	Pascal
Cloze Test	English	75.01	77.07
	Pascal	81.97	84.72

It is interesting to note that the lowest scores were obtained on the English cloze tests, presented after reading the same English programs, and that the highest scores were earned on the Pascal cloze tests, presented after reading the same Pascal programs. Exposure to the program in a different language during the same session improved the English cloze scores, but made the Pascal cloze scores worse. It appears that exposure to the Pascal faciltated the English cloze scores but exposure to the English programs seems to have had a negative influence on the Pascal cloze scores.

Differences Between Subjects

Even though the subjects had the same exposure to the programming languages and the algorithms, other attributes of the subjects were expected to influence their ability to comprehend programs. Measures of differences between subjects included: (a) whether or not they were native English speakers, (b) Pascal program production scores, (c) Previous programming experience, and (d) self-rated reading ability.

Native language

Table 26, in Appendix F, gives the scores by program and language for each dependent variable for the native and non-native English language categories. Table 16 shows the means by programming language of the seventeen native and six non-native English language subjects for each of the dependent variables For this experiment, all of the non-native English speakers had Chinese as their first language and they all spoke fluent English. Therefore, further study is needed before these results could be generalized.

Table 16

Scores of Native and Non-Native English Speakers by Computer Language

	English Speakers			
	Non-Native		Native	
	English	Pascal	English	Pascal
RMEM	61.33	57.00	59.63	56.67
CLOZE	69.32	78.11	81.44	83.73
SAC	29.21	30.89	20.20	20.18
SREAD	150.16	190.69	112.53	156.50

The non-native English language subjects took longer than the native English language subjects to correctly answer the RMEM questions (SAC). Possibly the non-native speakers do not have information efficiently stored in their memory. It may then take longer to retrieve the information from memory. The non-native English speakers also took longer to read the treatments. This result suggests that non-native English speaking subjects either are unable to retrieve information from the text directly or they need to extract more information from the text. It is also possible that they may have a sparser cognitive framework which could require more time to integrate information.

The cloze scores were also better for the native English language subjects. This result was expected particularly for the English programs since the cloze test is partially a measure of the subject's ability to use textual features to extract meaning. However, the non-native English speakers received a much greater difference in favor of the Pascal treatments than did the native English speakers for both the recognition memory (RMEM) and Cloze scores.

Pascal program production

The second measure of differences between subjects consisted of program production scores. The subjects wrote several short Pascal programs which were each given from zero to six points. The mean score of all the programs was 2.99. The subjects were divided into two groups. The above-mean group consisted of subjects with mean production scores greater than or equal to 3.00. The below-mean group was comprised of subjects less than the 2.99 average. None of the subjects had a score exactly equal to the average.

Table 27, in appendix F, consists of the mean scores by program for each of the dependent variables by Pascal production group and language. Table 17 shows the means by language of the below and above the mean groups for the dependent variables.

Table 17

Influence of Pascal Production Ability on Program Comprehension

	Below-Mean		Above-Mean	
	English	Pascal	English	Pascal
RMEM	56.46	50.00	61.88	60.33
CLOZE	69.29	78.86	80.16	84.53
SAC	26.04	24.43	20.08	21.48
SREAD	148.19	158.44	105.98	166.92

The high (above the mean) group performed better on the recognition memory (RMEM) tests. They also took less time to correctly answer the RMEM questions (SAC). The high group had better cloze scores for both languages, but the difference between the cloze scores of the two groups was greater for the English than it was for the Pascal programs. Suprisingly, the number of seconds the subjects took to answer the RMEM questions (SAC) and the number of seconds they took to read the treatments (SREAD) were less for the English programs but not for the Pascal programs. Although one might expect the Pascal treatments to be read and answered more quickly by the group which produces above average Pascal programs, this was not the case. Possibly

they studied the programs and thus, understood them better. It was also curious that this better production group read the English programs so quickly. Possibly, the faster times reading the English programs were because of the subjects who had read the Pascal version first and were already familiar with it.

Programming experience

The third measure of subject differences was the number of programs previously written in any computer language. The median number of programs written by all the subjects was 30. The subjects who wrote more than 30 programs were categorized as high-experience subjects. Those subjects who wrote fewer than 30 programs belonged to the low-experience group. Table 28, in Appendix F, gives the scores for the dependent variables by program, language and program experience group. Table 18 consists of the means by language for the high and low experience groups.

Table 18

Influence of Programming Experience on Program Comprehension

	Below-Mean		Above-Mean	
	English	Pascal	English	Pascal
RMEM	55.12	50.71	67.59	66.11
CLOZE	70.71	78.71	85.25	88.40
SAC	25.09	24.41	17.59	19.54
SREAD	129.58	162.56	106.80	166.16

The high-experience group performed better for the recognition memory (RMEM) and the cloze scores. The high-experience group also spent less time correctly answering the RMEM questions (SAC) than the low-experience group. The recognition memory (RMEM), cloze scores, and time to correctly answer the RMEM questions (SAC) were all better for the high experience group in both languages. The number of seconds taken to read the treatments (SREAD) was better (less) for the high-experience group when they read the English programs. This was not the case for the Pascal programs. Possibly the high- experience group used the Pascal programs to facilitate comprehension of the English programs.

Self-rated reading ability

The final measure of individual differences was the subject's reading ability. Subjects were asked to evaluate their reading ability on a five point scale. Although this measurement is subjective, perceived ability often reflects true ability. Subjects who rated their ability to read as good to excellent were placed in the high ability group while those who rated their ability as poor to fair were classified in the low ability group. Table 29, in Appendix F, consists of the mean scores for all the dependent variables by program, language, and reading ability group. Table 19 shows the means by language and ability group for the dependent variables.

Table 19

Influence of Reading Ability on Program Comprehension

	Below-Mean		Above-Mean	
	English	Pascal	English	Pascal
RMEM	51.87	44.79	64.33	63.11
CLOZE	71.20	75.88	79.59	85.93
SAC	19.22	17.98	23.72	24.92
SREAD	100.86	140.26	131.51	176.67

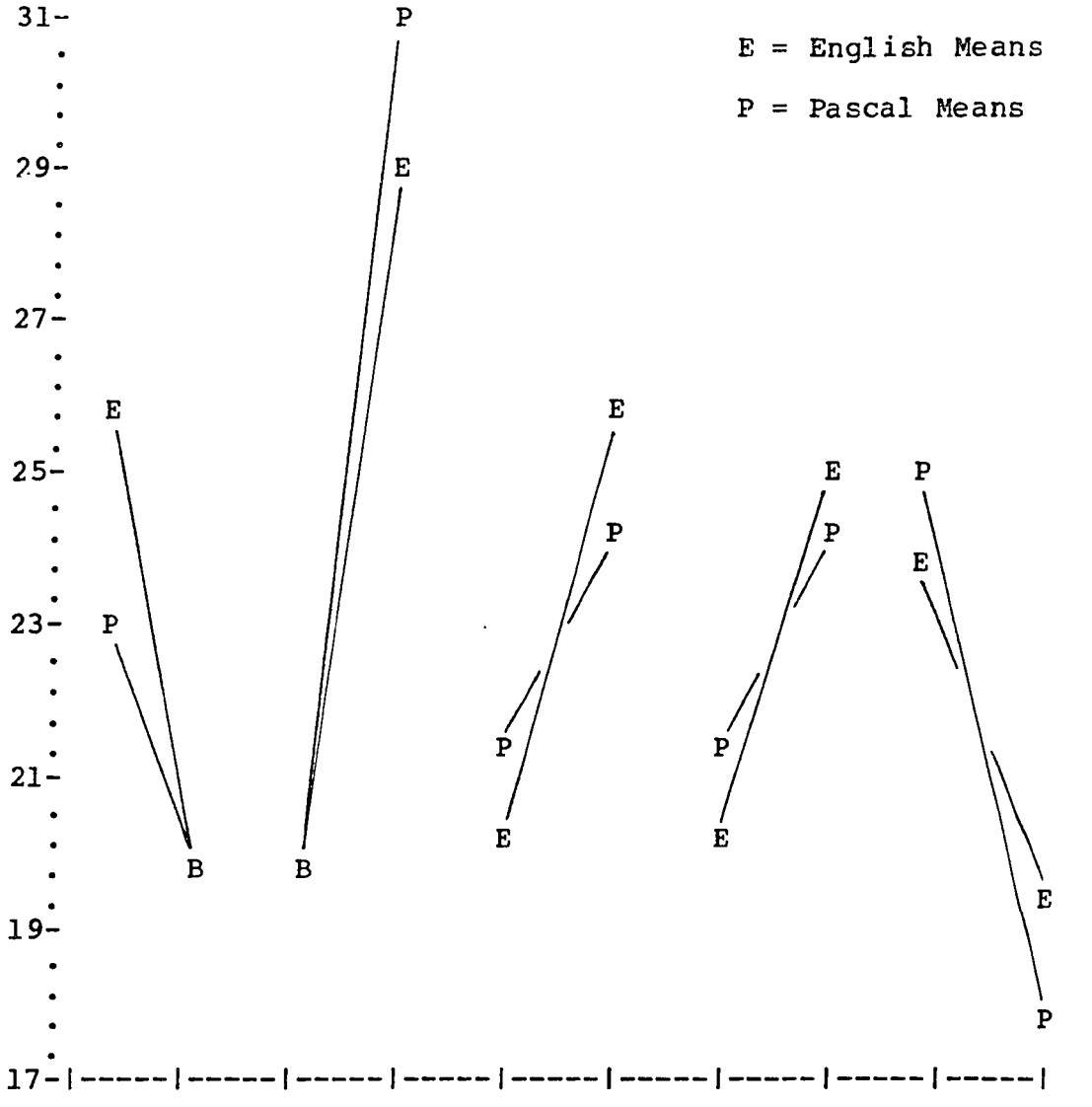
The self rated high ability readers received better recognition memory (RMEM) and cloze scores. The overall

means of the number of seconds that the high ability readers took to both read and correctly answer the RMEM questions was greater than that of the self-rated low ability readers. This may indicate that the skilled reader needs more time to store the information because there is more to store. Also it may takes more time for the skilled reader to make inferences because he has more information to process.

Combined differences between subjects

Examination of the collective individual difference scores reveals some counter-intuitive results. Higher recognition memory and cloze scores and lower response times were expected for the native English speakers and the above-the-mean groups in the other measures. The native English speakers and the above the mean groups of the other measures did better on the recognition memory and the cloze scores as was predicted. However, the response times for both the time to correctly answer the recognition memory questions (SAC) and the time to read the programs (SREAD) were not consistently better for the above the mean groups. Figure 10 illustrates the relationships between the languages and the subject difference groups. The extreme gist groups were included to show a comparison of the time to correctly answer for the subject differences and the knowledge of the program's purpose.

Figure 10. Seconds to Correctly Answer by Subject Difference group and Language.



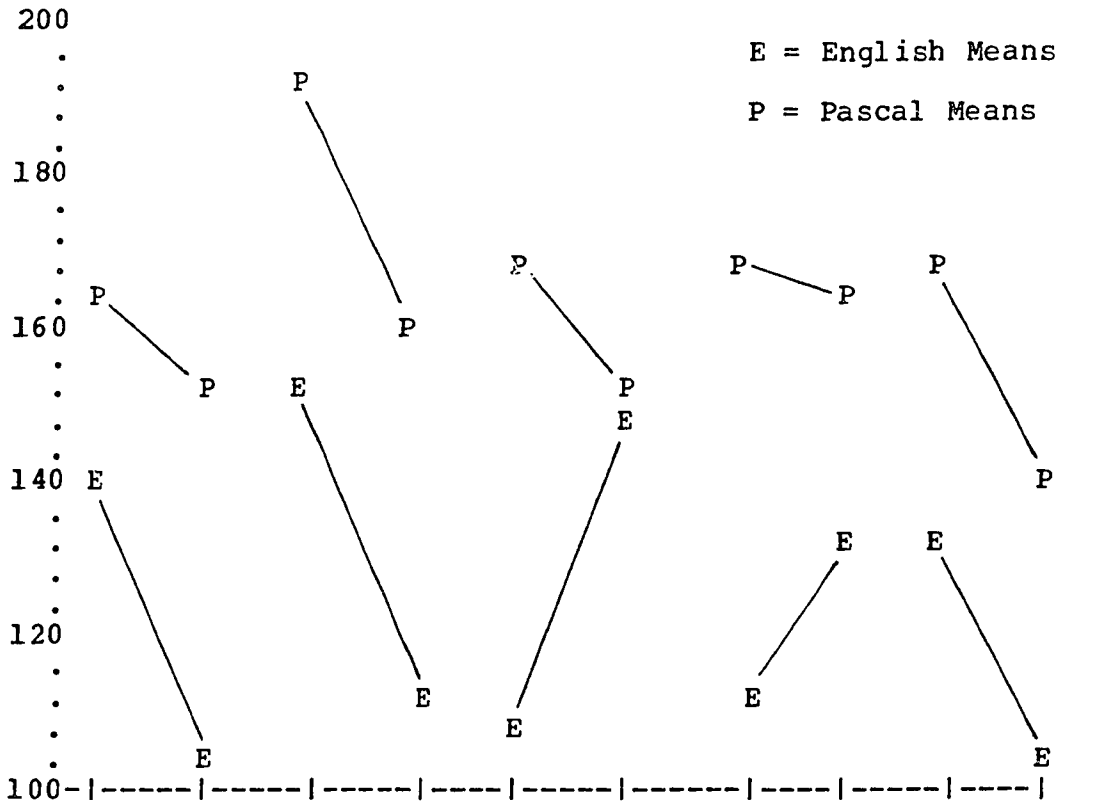
Notice that three of the group difference scores (native English speakers, Pascal production, and programming experience) show response times in the expected direction. The measures of self rated reading ability times are in the direction opposite from the expected one. A possible explanation for the longer recall times for the above average reading ability group is that self-rated scores are not accurate. An alternate explanation is that the poor readers answer faster because they are guessing and get some correct answers by chance. The low recognition memory and cloze scores for this group suggest that guessing is a possible strategy. Further support for the guessing hypothesis is the observation that the group which did not know the program's purpose in either language (N-Gist) took less time to answer correctly than did the group that understood the gist (B-Gist).

Pascal production, programming experience, and self rated reading ability all interacted with language. It is interesting to note that the below average group for the Pascal production and programming experience used less time to correctly answer questions about the Pascal programs, while the above average group for these two measures took less time to answer the questions for the English programs.

Figure 11 illustrates the number of seconds taken to read the programs by language and subject difference group.

Again the extreme gist groups are included so comprehension can be compared to response time.

Figure 11. Time to Read by Language and Subject Difference Group



Careful examination of Figure 11 shows several unexpected results. In all four of the subject difference measures, the Pascal programs took longer for the above

average groups to read. The native English speakers and the higher self rated reading ability group also took longer to read the English programs. Only the Pascal production and programming experience measures for the English programs are in the expected direction. This counter-intuitive result may have occured because the native speakers and the above the mean groups had more complex internal schemata so they might have needed more time to resolve their expectations. Possibly, the native speakers and the above average groups took more time to understand the material. They could have been more aware of the rules of the Pascal language and used those rules to facilitate program comprehension.

Discussion

Experiment two was designed to investigate the interaction of language type with characteristics of the subjects. It asked how the textual features, inherent in the language, and attributes of the subject interacted to influence program comprehension. The subject-related properties consisted of knowledge of the program's gist, experimentally manipulated training and language ability differences between the subjects.

Results from experiment two suggest that language type did interact with knowledge and experience of the subjects to influence comprehension. This conclusion appears to

conflict with the results of Dyck and Mayer (1985). Their study found that subjects needed longer response times for program statements written in English than in Basic and that there was no interaction between language and response times. Their response times were the times to read and answer questions on eight program statements written in English and BASIC. This variable was similar to a combination of the time taken to read (SREAD) and the time to answer recognition memory questions (SAC) used in experiment two. Although the response time variables in this study showed an interaction with language, the recognition memory scores (RMEM) and the cloze scores (CLOZE) were more sensitive to language type (English or Pascal).

The difference in the results could be explained by individual differences. Dyck and Mayer used different subjects to read each language rather than include language as a repeated measure for the same subjects. The group that read the statements written in English were not familiar with the programming language Basic. They may also have been unfamiliar with the concepts of the statements.

In addition to the possible subject differences, the response times for the English statements could have been greater because the English version was longer than the BASIC. BASIC is a simpler programming language than Pascal,

and BASIC programs are generally much shorter than either a Pascal or English version of the same thing.

The measurements of Dyck and Mayer were directed at program statements rather than at programs of varying complexity as was the case for experiment two. In the same sense that results of experiments on sentence comprehension differ from results of experiments on understanding passages, it could be expected that measurements of statement comprehension (Dyck and Mayer) might differ from program comprehension (this study).

A variable that proved to cause a significant interaction between language and features of the text, was the amount of nesting (HIER). If there were 0, 1, or 2 embedded IF-THEN-ELSE or REPEAT constructs in the program, the subjects received higher recognition memory scores (RMEM) for the English programs. On the other hand, if 3 or 4 of the constructs were embedded, the subjects received higher RMEM scores for the Pascal programs.

McQuitty's (1957) typal analysis was used to draw out the three dominant clusters. These consisted of: (a) lexical and propositional density, the number of lines and propositions, and the reading and retrieval time, (b) readability index and cloze scores, number of tokens and the level of nesting; and (c) the recognition memory and gist scores. When the correlations were separated by language,

it was possible to see that the means of the dependent variables for the Pascal programs were more correlated with textual features than were the means of the programs written in English. The level of nesting was correlated with the English recognition memory score but the time to read, time to recall and cloze scores were more correlated for the Pascal programs.

Knowledge of the subject interacted with language to influence program comprehension as was anticipated from the review of related research. If the subject knew the gist of the program, for both language treatments, they were considered to have a well developed schema for the program. Conversely, if they didn't know the gist for either language version of the program, they were thought to have a poorly developed schema for the program. The subjects thought to have well developed schemata, received higher recognition memory scores for the programs written in English. Subjects believed to have poorly developed schemata had higher recognition memory scores for programs written in Pascal.

The subject's prior knowledge was manipulated experimentally, both within a session and between the two sessions. The results from both of these conditions suggested that the Pascal programs provided a better framework in memory than the English programs for the second presentation of the treatments.

Differences between subjects were found in the areas of their language ability, programming production ability, and programming experience. Non-native English speakers were expected to have more difficulty with the English programs. This was found to be true particularly for the cloze scores. The non-native English speakers did not perform as well as the native English speakers on any measure, but their Pascal cloze scores were significantly better than their English cloze scores.

The subjects who averaged the highest Pascal production scores, and had the most prior experience received the highest recognition memory scores and cloze scores for both language versions of the programs. These groups also answered questions faster and spent less time reading the English programs.

The subjects with higher self-rated reading ability did better than the subjects with the lower self-rated reading ability on the recognition memory and the cloze tests. An unexpected finding from experiment two was that the Pascal programs took longer to read for the above average group for all of the subject difference measurements and the English programs took longer to read for the native English speakers and the self rated better readers. One possible explanation is that even when the above average subjects had little prior knowledge of the program they were

able to use their knowledge of the language to figure out what the program did. The time-to-answer measures for the subject differences were also unexpected. The below average subjects in Pascal production, programming experience, and self-rated reading ability had faster response times for programs written in Pascal than they did for Programs written in English. A look at the gist scores when the subjects knew the program's purpose in both languages (B-Gist) shows that the subjects correctly answered questions for the Pascal programs faster than they did for the English programs. Further inspection shows that if they did not know the gist at all (N-Gist) the questions were answered faster for both languages. There is a possibility that the faster SAC times for the N-Gist subjects were because they guessed the correct answer.

Results from experiment two suggest that there is a rather complex relationship between the language, textual features, subject's prior knowledge, training, and differences between the subjects which influences program understanding.

CHAPTER VI

Conclusion

Results from the two experiments suggest that attributes of the subject interact with features of the text to influence program comprehension. Experiment one was confined to one subject attribute (his knowledge of the program's purpose) and two types of textual features (propositional density and program explicitness). Binary measurements of program recall were categorized by the subject's knowledge of the program's purpose and the two types of propositional density and program explicitness. As was hypothesized, the knowledge of the program's purpose facilitated the recall of high propositional density programs. The subjects who were not aware of the program's purpose were of particular interest in this study since considerable programming experience is needed before prior knowledge of most programs can be expected. It was anticipated that program recall of subjects who were not aware of the program's purpose would be influenced by features of the text. If the subjects did not know the program's purpose, they could still successfully recall the low propositional density programs.

It was predicted that subjects who did not know the program's purpose should recall explicit programs better than implicit ones since the program's purpose could be

deduced from the algorithm. This prediction was not realized. The subjects who did not know the program's purpose were evenly divided between whether or not they successfully recalled the explicit programs. However, these same subjects recalled 2.2 times more implicit programs than they did not recall. This result was consistent with the natural language research of Anderson (1974). The better memory which subjects had for implicit text could be attributed to the fact that they had to actively derive the missing information. This additional processing facilitated program recall.

The methodology of experiment two was guided by the results of experiment one. In experiment one the propositional density was not separated from other features of the text such as lexical density, program size, and the amount of propositional hierarchy. Experiment two expanded the textual features to specifically include these variables.

The programs in experiment one were written in a typical computer language (ADA). Whether the use of a natural language as a programming language facilitates comprehension better than a conventional programming language has been the subject of debate among many computer scientists. The language factor for experiment two was extended to compare the interactions of textual features and

attributes of the subject in both a natural language (English) and a computer language (Pascal).

The subject attributes were increased for experiment two to include personal characteristics such as whether or not they were native English language speakers, above and below mean programming experience, above and below programming performance, and above and below mean self rated reading ability. Prior exposure to the program was experimentally manipulated both within and between reading sessions.

The subject's knowlege of the program's purpose became a dependent variable for experiment two. In addition the dependent variables for experiment two included recognition memory scores, cloze scores, and two response time measurements (time taken to read and average time to correctly answer the recognition memory questions). Although the response times were measures of processing time rather than comprehension, they were used in conjunction with the comprehension measures to lend insight into the various interactions of the independent variables.

The results of experiment two suggested that the the language, features of the text, and attributes of the subject all interact to influence program comprehension. It is not possible to state unequivocally that the subjects comprehend programs written in English better than they do

if they are written in Pascal, although that is sometimes the case. If the subjects are aware of the program's purpose or if the depth of the hierarchy is one or two, the subjects do receive higher recognition memory scores for programs written in English. However, If the subjects are not aware of the program's purpose or the depth of the hierarchy is three or four, the subjects receive higher recognition memory scores for programs written in Pascal.

The performance of the subjects who did not know the program's purpose is important since natural language programming languages have been advanced as being useful for a large segment of the population which is not likely to have a reservoir of a priori knowledge of programs and algorithms. In fact, it is probably the expert programmers, rather than beginning programmers, who are most likely to have schemata for programs which include prior experience with a particuar algorithm for a specific purpose.

The average depth of hierarchy used in most production programs is considerably higher than the three or four levels which comprised the larger depth of hierarchy in experiment two. Yet, it was more difficult for second semester programming students to understand the English than the Pascal versions of the more complex experimental programs. However, expert programmers may be able to chunk greater levels of hierarchy into units.

An interaction of language type and depth of hierarchy indicates that computer applications such as text editors which usually have sequential commands or one level of hierarchy might yield better performance if the commands were in English. It is also likely that users of a text editor would be very familiar with the purpose of the text editing command. Results from experiment two would again predict superior performance for a text editor which used English commands. This prediction in fact agrees with the experimental evidence of Roberts (1979).

This study suggests that a natural language may not be suitable as a programming language at least for non-expert programmers. One reason is that a natural language may not have the facility to unambiguously express programs with more than two levels of hierarchy. In addition, a well developed knowledge structure concerning the purpose of the program is not usually available before the program is read. The programming language has the advantage of having a limited set of specific rules which the subject can employ to determine what the program does. This knowledge can possibly then be used to extract the program's overall purpose. Even if the higher level knowledge of what the program does is not resolved, the subject can still ascertain most of the steps of the algorithm. A natural language programming language does not have a limited set of

specific rules. If the subject misreads or does not understand specific statements, there is not an intermediate level of understanding which can be used for inferences.

This study suggests that a typical computer programming language is superior to a natural programming language, at least for non-expert programmers when the knowledge of the program's purpose is not known or when the programs have more than two levels of hierarchy. Both of these conditions are usually true for the majority of application programs read by the non-expert programmer. Thus programs written using a typical computer language would probably be more readily "understood" in many situations. There is some research corroborating that this implication holds even for professional programmers. Simmons (1986) reported that M. Jarke et al. found subjects who used a natural language programming language to specify complex retrieval instructions from a data base were correct less than half as often as when they used the programming language SQL to specify the retrieval instructions. Small and Weldon (1983) also found subjects could perform data manipulation with the computer language SEQUEL faster and more accurately than they could with a natural language subset.

This research was conducted using second year programming subjects. Future research needs to expand the experience level of the subjects to include experts. This

is particularly true if the results are to be generalized to include professional programming applications.

In addition, the subject attributes need to be expanded. Instead of using the subject's self-rated ability, the accuracy of this measure should be increased by employing areading test. Additional knowledge about the relative importance of some textual features could be collected by tracking the subject's eye movements as they read programs. Eye movement data could be used to determine whether features of the program, such as the amount of hierarchy and the position of the propositions in the hierarchy level, were difficult to process.

Results from experiment one suggested that the explictness of the program was a salient variable in the effect of textual features on program comprehension. This factor was not included in experiment two. However, it should be employed in future research, particularly in experiments that record eye movements.

Results from experiment two imply that the program written in Pascal facilitated non-expert programmers comprehension of the same program written in English but the converse was not the case. This may be due to the fact that the subjects need to participate more actively in the comprehension process when they read programs written in a computer language. In turn, this activity may help develop

a schema for the program which is transfered to the next presentation of the same program. Again this implication needs to be investigated with a range of subjects which includes expert programmers. Experts are more likely to know more about a program's purpose than second semester programming students. A program written in English may well provide the schema for expert subjects on larger programs than were used in this study. The fact that in practice comments for programs are written in a natural language and are used to provide knowledge about the program's purpose tends to support this possibility. The natural language comments found in most programs, however, tend to be a series of statements. That is, they do not usually have several levels of hierarchy.

This study suggests that creating natural programming languages is not likely to facilitate the use of computers by beginning programmers. Possibly expert programmers will have a schema for the programs which will allow them to efficiently use natural programming languages. However, future research must be conducted which investigates this possibility.

APPENDIXES

APPENDIX A. ADA PROGRAMS USED IN EXPERIMENT ONE

PROGRAM #3 EXPLICIT ALGORITHM LOW PROPOSITIONAL DENSITY

Number of propositions 7
Number of tokens 49
Propositional Density 14%

TEXT:

```
function CHANGE-COURSE(D: DIRECTION: T:TURN)
                                        return DIRECTION is
   begin
     case T of
       when LEFT => return TURN_LEFT(D);
       when RIGHT =>return TURN-RIGHT(D);
       when STRAIGHT => return D;
       when REVERSE => return TURN_ABOUT(D);
     end case;
  end CHANGE_COURSE;
```

		GIST	
		yes	no
95%	yes	18	9
RECALL	no	2	3

PROGRAM #2 IMPLICIT ALGORITHM LOW PROPOSITIONAL DENSITY

Number of propositions 7
Number of tokens 36
Propositional Density 19%

TEXT:

```
    K: Short_INT := 0;
    J: SHORT_INT := 1:
  begin
    while K < 10000 loop
      PUT (K);
      K := J+K;
      J := K-J;
  end loop;
```

		GIST	
		yes	no
95%	yes	11	20
RECALL	no	0	1

PROGRAM #1 EXPLICIT ALGORITHM HIGH PROPOSITIONAL DENSITY

Number of propositions 9
Number of tokens 28
Propositional Density 32%

TEXT:

```
Max := V(1);
for I in 2..10 loop
  if V(I) > MAX then
      MAX := V(I);
end loop;
```

		GIST	
		yes	no
95%	yes	17	3
RECALL	no	2	10

P <= 0.0002

PROGRAM #4 IMPLICIT ALGORITHM HIGH PROPOSITIONAL DENSITY

Number of propositions 18
Number of tokens 51
Propositional Density 35%

TEXT:

```
if N mod 2 = 0 then
     PRIME := FALSE;
else
  FACTOR := 3;
  loop
     PRIME := TRUE;
     exit when FACTOR**2 > N;
     PRIME := FALSE;
     exit when N mod FACTOR = 0;
     FACTOR := FACTOR + 2;
  end loop;
end if;
```

		GIST	
		yes	no
95%	yes	14	6
RECALL	no	1	11

P <= 0.0009

APPENDIX B. PROGRAMS USED IN EXPERIMENT TWO

PASCAL PROGRAM SET TWO

PASCAL PROGRAM ID #9, MAXIMUM ELEMENT

Number of tokens = 104
Textual Density = 5%
Number of Propositions = 24
Propositional Density = 23%

TEXT:

```
 PROGRAM ONE (INPUT.OUTPUT);
  CONST ML= 100;
  TYPE LIST = ARRAY [1 .ML] OF REAL;
  VAR V    : LIST; L : INTEGER;
  FUNCTION A(V:LIST; L:INTEGER) :REAL;
    VAR K : INTEGER; X : REAL;
      BEGIN
        IF L > 0 THEN BEGIN
                 X := V[1];
                 FOR K :=.2 TO L DO
                   IF V[K] > X THEN X := V[K]
               END
            ELSE WRITELN(' PARAMETER ERROR');
       A := X
       END;
BEGIN
 L := 1;
 WHILE NOT EOF DO
   BEGIN
     READLN (V[L]); IF NOT EOF THEN L := L + 1;
   END;
 WRITELN('THE VALUE OF X IS EQUAL TO '.A(V,L));
END.
```

PASCAL PROGRAM ID #10, MEDIAN

Number of tokens = 89
Textual Density = 5%
Number of Propositions = 22
Propositional Density = 25%

TEXT:

```
10 PROGRAM TWO(INPUT-OUTPUT);
    CONST ML = 100;
    TYPE  LIST = ARRAY[1 .ML] OF REAL;
    VAR X : LIST;
        N : INTEGER;
    FUNCTION M( X:LIST; N:INTEGER) : REAL;
      BEGIN
        IF N < 1 THEN M := 0
           ELSE IF ODD (N)
               THEN M := X[(N+1) DIV 2]
               ELSE M := (X[N DIV 2] + X[N DIV 2 + 1]) / 2.0
      END;
   BEGIN
     N := 1;
     REPEAT
       READ (X[N]);
       N := N + 1;
     UNTIL EOF;
     WRITELN ('M=',M(X,N-1))
   END.
```

PASCAL PROGRAM ID #7, SQUARE ROOT

Number of tokens = 65
Textual Density = 3%
Number of Propositions = 14
Propositional Density = 22%

TEXT:

```
7 PROGRAM FOUR (INPUT,OUTPUT);
   VAR X : REAL;
   FUNCTION Q(X:REAL) : REAL;
     VAR R,S : REAL;
     BEGIN
       R := X;
       S := 1;
       WHILE ABS (S - R) > 0 000001 DO
         BEGIN
           R := S;
           S := (R * R + X) / (2 0 * R)
     END;
     Q := S
   END;
  BEGIN
   WHILE NOT EOF DO
     BEGIN
       READLN(X);
      WRITELN(X,Q(X))
     END
  END.
```

PASCAL PROGRAM ID # 1, GREATEST COMMON DIVISOR

Number of tokens = 68
Textual Density = 1%
Number of Propositions = 24
Propositional Density = 35%

TEXT:

```
1 PROGRAM FIVE(INPUT,OUTPUT);
   VAR N1,N2,N3 : INTEGER;
   PROCEDURE M( I,J :INTEGER; VAR K:INTEGER);
     BEGIN
       K := I;
       WHILE ( K >= J) DO
         K := K - J
     END;
  BEGIN
   WHILE NOT EOF DO
     BEGIN
       READLN (N2,N3);
       WRITE (N2,N3);
       WHILE N3 <> 0 DO
         BEGIN
           N1 := N2;
           N2 := N3;
           M(N1,N2,N3)
         END;
       WRITE (' M IS', N2)
     END
  END.
```

PASCAL PROGRAM ID # 8, MEAN

Number of tokens = 93
Textual Density = 4%
Number of Propositions = 19
Propositional Density = 20%

TEXT:

```
8 PROGRAM SEVEN (INPUT,OUTPUT);
   CONST ML = 100;
   TYPE LIST = ARRAY [ 1..ML] OF REAL;
   VAR V : REAL; L : INTEGER;
   FUNCTION M( X:LIST; L:INTEGER) : REAL;
     VAR I : INTEGER; S : REAL;
    BEGIN
      IF L > 0 THEN BEGIN
           S := 0;
           FOR I := 1 TO L DO
              S := S + X[I];
           M := S / L
         END
       ELSE M := 0
    END;
   BEGIN
    L := 1;
    WHILE NOT EOF DO
     BEGIN
       READ (V[L]); L := L + 1
     END;
   WRITELN ( 'M=', M(V,L-1))
  END.
```

PASCAL PROGRAM ID #12, PRIME NUMBER

Number of tokens = 111
Textual Density = 4%
Number of Propositions = 30
Propositional Density = 27%

TEXT:

```
12 PROGRAM TWELVE (INPUT,OUTPUT);
    VAR MX, X : INTEGER;
    FUNCTION P (N : INTEGER) : BOOLEAN;
     VAR FACTOR : INTEGER; FLAG : BOOLEAN;
    BEGIN
    IF N <= 1 THEN P := FALSE
      ELSE  IF N = 2 THEN P := TRUE
      ELSE IF NOT ODD(N) THEN P := FALSE
      ELSE BEGIN
             FLAG := TRUE;
             FACTOR := 3;
             WHILE (FLAG) AND (N > FACTOR * FACTOR) DO
             IF N MOD FACTOR <> 0 THEN FACTOR := FACTOR + 2
                                  ELSE FLAG := FALSE;
             P := FLAG
           END
   END;
 BEGIN
   READLN (MX);
   WRITELN (' THE TRUE VALUES FROM 1 TO ',MX,' ARE:');
   FOR X := 1 TO MX DO
     IF P(X) THEN WRITE(X)
 END.
```

PASCAL PROGRAM ID #5, DIRECTION

Number of tokens = 75
Textual Density = 4%
Number of Propositions = 26
Propositional Density = 35%

TEXT:

```
5 PROGRAM THREE (INPUT,OUTPUT);
   TYPE DIRECTION = (LEFT,RIGHT,BACK,STRAIGHT);
   VAR TURN : DIRECTION;
          D : INTEGER;
          T : CHAR;
   BEGIN
     READLN(T,D);
     IF T = 'L' THEN TURN := LEFT
       ELSE IF T = 'R' THEN TURN := RIGHT
       ELSE IF T = 'B' THEN TURN := BACK
       ELSE TURN := STRAIGHT;
     CASE TURN OF
       LEFT  : WRITELN ( 'TURN LEFT',D);
       RIGHT : WRITELN ( 'TURN RIGHT',D);
       STRAIGHT : WRITELN ( 'FORWARD',D);
       BACK : WRITELN ( 'BACKWARD',D)
     END
  END.
```

PASCAL PROGRAM ID # 6, EXPONENTIAL EQUATION

```
     Number of tokens = 54
     Textual Density  = 4%
     Number of Propositions = 11
     Propositional Density = 20%

  TEXT:

 6 PROGRAM SIX(OUTPUT);
    VAR X,S,L : REAL;
    BEGIN
      X := 0.5;
      S := 0.25;
      L := 1.0E-6;
    REPEAT
      IF EXP(X) < 3 * X
         THEN X := X - S
         ELSE X := X + S;
         S := S / 2.0
    UNTIL S < L;
    WRITELN ('X =', X:8:6)
   END.
```

PASCAL PROGRAM ID # 2, RANDOM NUMBER GENERATOR

```
     Number of tokens = 60
     Textual Density  = 4
     Number of Propositions = 13
     Propositional Density = 22%

  TEXT:

 2 PROGRAM EIGHT (INPUT,OUTPUT);
    CONST N = 100;
    VAR   S, I : INTEGER;
    FUNCTION R (VAR S:INTEGER) : REAL;
     CONST A = 5913;
           C = 1;
           M = 65536;
     BEGIN
      S := (A * S + C) MOD M;
      R := S / M
     END;
   BEGIN
    READLN (S);
    FOR I := 1 TO N DO
      WRITELN (' R=',R(S))
   END.
```

PASCAL PROGRAM ID # 3, FACTORIAL

```
     Number of tokens = 48
     Textual Density  = 3%
     Number of Propositions = 11
     Propositional Density = 23%

  TEXT:

 3 PROGRAM NINE (INPUT,OUTPUT);
    VAR X,N : INTEGER;
    FUNCTION F(N:INTEGER) : INTEGER;
      BEGIN
        IF N <= 1
           THEN F := N
           ELSE F := N * F(N - 1)
      END;
   BEGIN
    WHILE NOT EOF DO
      BEGIN
       READLN (X);
       WRITELN ('F=',F(X))
      END
   END.
```

PASCAL PROGRAM ID # 4, FIBONACCI SERIES

```
     Number of tokens = 38
     Textual Density  = 3%
     Number of Propositions = 9
     Propositional Density = 24%

  TEXT:

 4 PROGRAM TEN (INPUT,OUTPUT);
    VAR K,J : INTEGER;
    BEGIN
      J := 0;
      K := 1;
      WRITELN ('K=');
      WHILE (K < 1000) DO
      BEGIN
        WRITELN (K);
        K := J + K;
        J := K - J
      END
  END.
```

PASCAL PROGRAM ID #11, FREQUENCY

```
     Number of tokens = 94
     Textual Density  = 5%
     Number of Propositions = 19
     Propositional Density = 20%

  TEXT:

11 PROGRAM ELEVEN(INPUT,OUTPUT);
    CONST MN = 255;
    TYPE LIST = ARRAY [0..MN] OF INTEGER;
    VAR L,N,T : INTEGER;
            F : LIST;
    BEGIN
      FOR L := 0 TO MN DO
        F[L] :=  0;
    READ (T);
    REPEAT
      IF N <> T THEN BEGIN READ (N);
         IF  (N >= 0) AND (N <= MN) THEN F[N] := F[N] + 1
       END;
    UNTIL N = T;
    FOR L := 0 TO MN DO
      BEGIN
        IF F[L] <> 0 THEN WRITELN(' VALUE IN ',L,' IS',F[L])
      END
   END.
```

ENGLISH PROGRAM ID #9, MAXIMUM ELEMENT

Number of tokens = 142
Textual Density = 12%
Number of Propositions = 18
Propositional Density = 13%

TEXT:

This program uses values in an array called V. The number of these values are put into a variable L. The function A uses the value of V and L. There are three conditions which depend on the value of L. If L is less than one, a message, "parameter error", is written. Otherwise, the first element of V is put into a temporary variable called X. If L is equal to one, the value of X becomes the answer. If L is greater than one, successive comparisons are made. Beginning with the second element, every element of V is examined and compared to the current value contained in X. Each time the element is greater than X, it replaces the current value of X. After all the comparisons have been made, the value that is currently in X becomes the answer.

ENGLISH PROGRAM ID #10, MEDIAN

Number of tokens = 126
Textual Density = 14
Number of Propositions = 10
Propositional Density = 8%

TEXT:

This program uses a function named M. M is given a set of ordered values which have been placed in an array called X. The number of the values is put into a variable called, N. If N is less than one, the value of the function M becomes zero. Otherwise, if N is odd then M becomes the value which corresponds to the position number obtained by dividing two into N and adding one. If N is not odd, then M becomes the value obtained by dividing two into the sum of the values that correspond to the position N divided by two and the position N divided by two plus one.

ENGLISH PROGRAM ID #7, SQUARE ROOT

Number of tokens = 119
Textual Density = 13%
Number of Propositions = 11
Propositional Density = 9%

TEXT:

A function Q uses a predefined variable X. The variable, R, is made equal to X. The initial value of a variable, S, is made equal to one. A test is made to see if the absolute value of S minus R is within a .000001 tolerence. If it isn't, the following actions are repeated: first R is made equal to the current value of S, then R is squared and added to X, next this sum is divided by two times R, this quotient then becomes the new value of S. When S is within the required .000001 units of the value of R, the procedure ends with the value of the procedure obtained from the variable, S.

ENGLISH PROGRAM ID #1, GREATEST COMMON DIVISOR

Number of tokens = 122
Textual Density = 14%
Number of Propositions = 17
Propositional Density = 14%

TEXT:

In a program that has variables N2 and N3 initially set, the following events happen as long as there is data: The value of N1 is made equal to the value of N2 and the value of N2 becomes N3. An internal procedure M, uses N1, N2, and N3 by the names of I, J, and K, as follows: While K is greater than or equal to J, J is subtracted from K and that difference becomes the next value of K. As soon as the value of K is less than the value of J the procedure ends with the result left in the variable, K. This procedure is used until N3 equals zero. Then the variable N2 contains the answer.

NATURAL LANGUAGE SENTENCE ID #8, MEAN

Number of tokens = 84
Textual Density = 12%
Number of Propositions = 14
Propositional Density = 17%

TEXT:

There is function, M, which uses a set of values V, and L, the number of the values in the set. S is initially equal to zero. If L is less than or equal to zero, an error is indicated and the function, M is set to zero Otherwise no error indication is given, each element of the set is added to the accumulative sum of S, and the value of S, the answer of procedure M, is obtained by dividing L into S.

ENGLISH PROGRAM ID #12, PRIME NUMBER

Number of tokens = 162
Textual Density = 14%
Number of Propositions = 22
Propositional Density = 14%

TEXT:

In order to determine if a function P has a true or false value, the following tests are given: If N is less than two, the value of P is false. If N is equal to two, the value of P is true. However, if N is any other even number, the value of P is false. For any other N the following steps are taken: A variable named Factor is set equal to three and P is initially given a value of true. As long as the value of P is true and N is greater than factor squared, the following test is repeated: If the remainder of N divided by Factor is not equal to zero, the value of Factor is increased by two. If the remainder is equal to zero, then the value of P becomes false. The value of P will remain true only when the remainder of N divided by factor doesn't become zero during the test.

ENGLISH PROGRAM SET TWO

ENGLISH PROGRAM ID #5, DIRECTION

Number of tokens = 119
Textual Density = 12%
Number of Propositions = 22
Propositional Density = 19%

TEXT:

The value of the variable Turn is established in this program from the first letter of the a variable called T. Also a number D is given, which specifies the quantity of Turn. If T is the letter L, then Turn becomes Left. If T is the letter R then Turn becomes Right. If T is equal to the letter B then Turn becomes Back. If T has any other value Turn becomes Straight. In the case where Turn is equal to Left, the message "Turn Left" D is written. If Turn equals Right, the message is "Turn Right" D. If Turn equals Back, the message is "Backward" D. However if Turn equals Straight, the message written is "Forward" D.

ENGLISH ID #6, EXPONENTIAL EQUATION

Number of tokens = 100
Textual Density = 14%
Number of Propositions = 11
Propositional Density = 11%

TEXT:

The variable X is given the value of one half, the variable S is given the value of one fourth, and the variable L is given the value one hundred thousanth. The following events are repeated until the value of S is less than the value of L. If e raised to the power X is less than three times X, X becomes X minus S. Otherwise the value of X becomes the sum of X plus S and S is divided in half. When S becomes less than or equal to L, the current value of X is written.

ENGLISH PROGRAM ID #2, RANDOM NUMBER GENERATOR

Number of tokens = 92
Textual Density = 12%
Number of Propositions = 10
Propositional Density = 11%

There is a program that initially gives a value S to a function R. The function R multiplies S by a prime number and adds the product to one. This sum then becomes the new value of S. S is then divided by the largest integer, and the remainder becomes the new value of S. S is divided by the largest integer value again and the quotient becomes the answer to the function, R. This function is repeatedly used to get new values of S within the range of zero to one.

PROGRAM ID #3, FACTORIAL

Number of tokens = 78
Textual Density = 13%
Number of Propositions = 9
Propositional Density = 12%

TEXT:

Function F is originally given a variable N. When N is less than or equal to one, the process ends and the value of N is returned. Each time F is used, N is made equal to N minus one. When N is greater than one, the value returned from function F will be the product of N times the result of using function F with the new value of N minus one assigned as its given variable.

ENGLISH PROGRAM ID #4, FIBONACCI SERIES

Number of tokens = 50
Textual Density = 13%
Number of Propositions = 9
Propositional Density = 18%

TEXT:

J is initially made equal to zero and K is made equal to one. A title, "K", is written. As long as K is less than one thousand the following events are repeated: the value of K is written, K becomes K plus J, and J becomes K minus J.

ENGLISH PROGRAM ID #11, FREQUENCY

Number of tokens = 162
Textual Density = 14%
Number of Propositions = 22
Propositional Density = 14%

TEXT:

There is a program which sets an array of numbers, F, equal to zero. There are several numbers which are checked to see if they are within the range of F, if a number is within the range, one is added to the position of the number in the array F. As soon as all the data is checked, all the numbers in the array F which are greater than zero are written.

APPENDIX C. PROPOSITIONAL ANALYSIS OF PROGRAMS

Propositional Description of Selected Pascal Statements

Statement Type	Propositional Form
Assignment	SET[VARIABLE,VALUE]
If	IF[COND,ACTION,ALTERNATE ACTION]
Repeat	DO [VAR,FROM VALUE, TO VALUE] WHILE [COND] UNTIL [COND]
Moveout	WRITE [VARIABLE-LIST]
Procedure Call	CALL [PROCEDURE-NAME,ARGUMENT-LIST]
Initialize data types	INIT[VARIABLE-TYPE]
Begin statement group	BEGIN
End statement group	END

PASCAL PROGRAM ID #9, MAXIMUM

Number of tokens = 104
Number of Propositions = 24
Propositional Density = 23%

PROPOSITIONAL DESCRIPTION:

```
 1 INIT [ML,100,(2)[SIZE-OF-ARRAY,ML,(3)[NAME-OF-ARRAY,V]
 4 SET [L,1]
 5 REPEAT [WHILE, (6), (7)]
 6 NOT EOF
 7 MOVEIN [V(L)]
 8 INCREMENT,L,1
 9 IF [(6),(8),(11)]
10 END [5]
11 MOVEOUT ["value of x is equal to",(12)]
12 CALL [A,V,L]
13 L > 0
14 IF [(13),15,21]
15 SET [X,V(1)]
16 REPEAT [K,2,L,20]
17 IF [(18),(19)]
18 V(K) > X
19 SET [X,V(K)]
20 INCREMENT [K]
21 END [(16)]
22 MOVEOUT ["parameter error"]
23 SET [A,X]
24 RETURN[(12)]
```

PASCAL PROGRAM ID #10, MEDIAN

Number of Tokens = 89
Number of Propositions = 22
Propositional Density = 25%

PROPOSITIONAL DESCRIPTION:

```
 1 INIT [ML,100]
 2 INIT [SIZE-OF-ARRAY,ML]
 3 INIT [NAME-OF-ARRAY]
 4 SET [N,1]
 5 REPEAT[UNTIL(EOF),9]
 7 MOVEIN [X[N]]
 8 INCREMENT [N,1]
 9 END [5]
10 MOVEOUT ["M=",(11)]
11 CALL [M,X,N]
12 SET [N, VALUE [N-1]]
13 IF [(14),(15),(16)]
14 N < 1
15 SET [M,0]
16 IF [(17), SET [M,(18),(19)]
17 N = ODD
18 VALUE [X[N/2+1]]
19 IF [ NOT(17),(21)]
20 RETURN [M]
21 SET [M, VALUE [(X[N/2]+X[N/2+1])/2]]
22 RETURN [M]
```

PASCAL PROGRAM ID #7, SQUARE ROOT

Number of tokens = 65
Number of Propositions = 14
Propositional Density = 22%

PROPOSITIONAL DESCRIPTION:

```
 1 WHILE [(2),(3)
 2 NOT [EOF]
 3 MOVEIN [X]
 4 MOVEOUT [X,5]
 5 CALL [Q,X]
 6 SET [R,X]
 7 SET [S,1]
 8 WHILE [IF [(9),(10),(11)]
 9 |S-R| > 0.000001
10 SET [R,S]
11 SET [S,(12)]
12 VALUE [(SQUARED[R]+X)/(2*R)]
13 SET [Q,S]
14 RETURN [Q]
```

PASCAL PROGRAM ID #1, GREATEST COMMON DIVISOR

Number of tokens = 68
Number of Propositions = 24
Propositional Density = 35%

PROPOSITIONAL DESCRIPTION:

```
 1 WHILE [(2),(3)]
 2 NOT [EOF]
 3 MOVEIN [N2,N3]
 4 MOVEOUT [N2,N3]
 5 WHILE [(6),(7),(23)]
 6 N3 <> 0
 7 SET [N1,N2]
 8 SET [N2,N3]
 9 CALL [M,N1,N2,N3]
10 INIT [I,N1]
11 INIT [J,N2]
12 INIT [K,N3]
13 SET [K,I]
14 REPEAT [(15),(16)]
15 K >= J
16 SET [J, VALUE [J-K]]
17 RETURN
18 SET [N3,K]
19 SET [N2,J]
20 SET [N1,I]
21 END [14]
22 GOTO [24]
23 MOVEOUT ["M IS ",N2]
24 END [5]
```

PASCAL PROGRAM ID # 8, MEAN

Number of Tokens = 93
Number of Propositions = 19
Propositional Density = 20%

```
 1 INIT [VAR[ML,100]]
 2 INIT [NAME-OF-ARRAY,L,ML]
 3 SET [L,1]
 4 WHILE [NOT EOF, 5]
 5 MOVEIN [V[L]]
 6 INCREMENT [L]
 7 END[4]
 8 MOVEOUT ['M=', 9]
 9 CALL [M,V,L-1]
10 INIT [X,V]
11 INIT [L,L-1]
12 IF [L>0,13,14]
14 SET[5,0]
15 REPEAT [I,1,L,18]
16 SET [S, VALUE[S+X(I)]]
17 SET [M, VALUE[S/L]]
18 INCREMENT I
19 RETURN [M]
```

PASCAL PROGRAM ID #12, PRIME

Number of Tokens = 111
Number of Propositions = 30
Propositional Density = 27%

```
 1 MOVE IN [MX]
 2 MOVEOUT ['THE VALUES FROM 1 TO',MX,' ARE']
 3 REPEAT [1,X,1,MX]
 4 IF [(5),(1),(29)]
 5 CALL (P,X)
 6 TRUE [P]
 7 INIT [N,X]
 8 IF [(9),(10),(14)]
 9 TRUE [N <= 1]
10 SET [FALSE[P]],(14)]
11 IF [(12),(13),(14)]
12 TRUE [N = 2]
13 SET [TRUE [P]],(14)]
14 IF [(15),(16),(17)]
15 NOT [ODD[N]]
16 FALSE[P]
17 TRUE [FLAG]
18 SET [FACTOR,3]
19 WHILE [(20) AND (21)]
20 TRUE [FLAG]
21 TRUE [N > FACTOR]
22 IF [(23),(24),(25)]
23 REMAINDER [N/FACTOR]
24 SET [FACTOR, FACTOR + 2]
25 FALSE[FLAG]
26 SAME [(19)]
27 SET [P,FLAG]
28 RETURN[P]
29 MOVEOUT[X]
30 INCREMENT [X]
```

PASCAL PROGRAM ID # 5, DIRECTION

Number of Tokens = 75
Number of Propositions = 26
Propositional Density = 35%

```
 1 INIT [TURN, (2)]
 2 INIT [DIRECTION, VALUE [LEFT,RIGHT,BACK, STRAIGHT]
 3 MOVEIN [T,D]
 4 IF [(5),(6),(7)]
 5 TRUE [T='L']
 6 SET [TURN,LEFT],
 7 IF [(8),(9),(10)]
 8 TRUE [T='R']
 9 SET [TURN,RIGHT]
10 IF [(11),(12),(13)]
11 TRUE [T='B']
12 SET [TURN,BACK]
13 SET [TURN,STRAIGHT]
14 IF [(15),(16),(26)]
15 TRUE [TURN='LEFT']
16 MOVEOUT ['TURNLEFT',D]
17 IF [(18),(19),(26)]
18 TRUE [TURN='RIGHT']
19 MOVEOUT ['TURNRIGHT',D]
20 IF [(21),(22),(26)]
21 TRUE [TURN='STRAIGHT']
22 MOVEOUT ['FORWARD',D]
23 IF [(24),(25)]
24 TRUE [TURN='BACK']
25 MOVEOUT ['BACKWARD',D]
26 END
```

PASCAL PROGRAM ID # 6, EXPONENTIAL

Number of Tokens = 54
Number of Propositions = 11
Propositional Density = 20%

```
 1 SET [X,0.5]
 2 SET [S,0.25]
 3 SET [L,1.0E-6]
 4 REPEAT [(9)]
 5 IF [(6),(7),(8)]
 6 TRUE [EXP(X)<3X]
 7 SET [X,X-S]
 8 SET [X,X+S]
 9 SET [S,S/2.0]
10 TRUE [S<L]
11 MOVEOUT ['X= ',X]
```

PASCAL PROGRAM ID # 2

Number of Tokens = 60
Number of Propositions = 13
Propositional Density = 22%

```
 1 INIT [N,100]
 2 MOVEIN [5]
 3 REPEAT [I,1,N,13]
 4 MOVEOUT ['R=',(5)]
 5 CALL [R,S]
 6 INIT [A,5913]
 7 INIT [C,1]
 8 INIT [M,65536]
 9 SET [S,(10)]
10 MOD [(S*A+C),M]]
11 SET [R,S/M]
12 RETURN [3]
13 INCREMENT [I]
```

PASCAL PROGRAM ID # 3, FACTORIAL

Number of Tokens = 48
Number of Propositions = 11
Propositional Density = 23%

```
 1 REPEAT [(10)]
 2 MOVEIN [X]
 3 MOVEOUT ['F',(4)]
 4 CALL (F,X)
 5 IF [(6),(7),(8)]
 6 TRUE [N<=1]
 7 SET [F,N]]
 8 SET [F,N*(9)]
 9 CALL [F,N-1]
10 UNTIL [TRUE[EOF]]
11 RETURN [4]
```

PASCAL PROGRAM ID # 4, FIBONACCI

Number of Tokens = 38
Number of Propositions = 9
Propositional Density = 24%

```
 1 SET [J,0]
 2 SET [K,1]
 3 MOVEOUT ['K=']
 4 REPEAT [(5)]
 5 WHILE [K<1000]
 6 MOVEOUT [K]
 7 SET [K,J+K]
 8 SET [J,K-J]
 9 END [4]
```

PASCAL PROGRAM ID #11, FREQUENCY

Number of Tokens = 94
Number of Propositions = 19
Propositional Density = 20%

```
 1 INIT [MN,255]
 2 INIT [ARRAY[F],MN]
 3 REPEAT [L,0,MN]
 4 SET [F[L],0]
 5 END [(3)]
 6 MOVEIN [T,N]
 7 REPEAT [(13)]
 8 IF [(9),(10),(11)]
 9 AND [(10),(11)]
10 TRUE [N>0]
11 TRUE [N<MN]
12 SET [INCREMENT [F(N)]]
13 UNTIL [N=T]
14 END [7]
15 REPEAT [L,0,MN]
16 IF [(17),(18)]
17 TRUE [T(L)=0]
18 MOVEOUT ['VALUE IN ',L, 'IS',F(L)]
19 END [(15)]
```

ENGLISH PROGRAM ID # 9, MAXIMUM

Number of Tokens = 142
Number of Propositions = 18
Propositional Density = 13%

PROPOSITIONAL DESCRIPTION:

```
 1 INIT [NAME-OF-ARRAY, V]
 2 SET [L,N]
 3 CALL [A,V,L]
 4 IF [(5),(6),(7)]
 5 L < 1
 6 MOVEOUT["parameter error"]
 7 SET [X,V(1)]
 8 IF [(9),(10)]
 9 L = 1
10 SET [ANSWER = X]
11 IF [(12),(13)]
12 L > 1
13 REPEAT [(14)]
14 WHILE, L,2,N
15 IF [(16),(17)]
16 V(L) > X
17 SET [X(L)]
18 END [14]
```

ENGLISH PROGRAM ID #10, MEDIAN

Number of tokens = 126
Number of Propositions = 10
Propositional Density = 8%

PROPOSITIONAL DESCRIPTION:

```
 1 CALL [M,X,N]
 2 IF [(3),(4),(5)]
 3 N < 1
 4 SET [M,0]
 5 IF [(6), SET [M,(7),(8)]
 6 N = ODD
 7 VALUE [X[N/2+1]]
 8 IF [ NOT(6),(9)]
 9 SET [M, VALUE [(X[N/2]+X[N/2+1])/2]]
10 RETURN [M]
```

ENGLISH PROGRAM ID #7, SQUARE ROOT

Number of tokens = 119
Number of Propositions = 11
Propositional Density = 9%

PROPOSITIONAL DESCRIPTION:

```
 1 MOVEIN [X]
 2 CALL [Q,X]
 3 SET [R,X]
 4 SET [S,1]
 5 IF [(6),(7),(10)]
 6 |S-R| > 0.000001
 7 SET [R,S]
 8 SET [S,(9)]
 9 VALUE [(SQUARED[R]+X)/(2*R)]
10 SET [Q,S]
11 RETURN [Q]
```

ENGLISH PROGRAM ID #1, GREATEST COMMON DIVISOR

Number of tokens = 122
Number of Propositions = 17
Propositional Density = 14%

PROPOSITIONAL DESCRIPTION:

```
 1 MOVEIN [N2,N3]
 2 WHILE [(3),4]
 3 N3 <> 0
 4 SET [N1,N2]
 5 SET [N2,N3]
 6 CALL [M,N1,N2,N3]
 7 INIT [I,N1]
 8 INIT [J,N2]
 9 INIT [K,N3]
10 REPEAT [(11),(12)]
11 WHILE (K>J)
12 SET [J, VALUE [J-K]]
13 SET [N3,K]
14 SET [N2,J]
15 SET [N1,I]
16 END [2]
17 MOVEOUT [N2]
```

ENGLISH PROGRAM ID # 8, MEAN

Number of Tokens = 84
Number of Propositions = 14
Propositional Density = 17%

```
 1 CALL [M,V,L]
 2 SET [S,0]
 3 IF [(5),(12)]
 4 L>0
 5 REPEAT [I,1,L,8]
 6 SET [S, VALUE[S+V(I)]]
 7 INCREMENT [I]
 8 END [5]
 9 SET [M,(10)]
10 VALUE [S/L]
11 GOTO [14]
12 SET [M,0]
13 MOVEOUT ['ERROR']
14 RETURN [M]
```

ENGLISH PROGRAM ID #12, PRIME

Number of Tokens = 162
Number of Propositions = 22
Propositional Density = 14%

```
 1 CALL (P,X)
 2 IF [(3),(4)]
 3 TRUE [N <= 2]
 4 FALSE[P]
 5 IF [(6),(7)]
 6 TRUE [N = 2]
 7 TRUE [P]
 8 IF [(9),(10)]
 9 TRUE [N>2]
10 FALSE [P]
11 TRUE [P]
12 WHILE [(13) AND (14)]
13 TRUE [FLAG]
14 TRUE [N > FACTOR]
15 IF [(16),(17),(18)]
16 REMAINDER [N/FACTOR]
17 FALSE[P]
18 SET [FACTOR, FACTOR + 2]
19 END [(12)]
20 RETURN[P]
21 IF [TRUE [P]]
22 MOVEOUT[X]
```

ENGLISH PROGRAM ID # 5, DIRECTION

Number of Tokens = 119
Number of Propositions = 22
Propositional Density = 19%

```
 1 MOVEIN [T,D]
 2 IF [(3),(4),(5)]
 3 TRUE [T='L']
 4 SET [TURN,LEFT],
 5 IF [(6),(7),(8)]
 6 TRUE [T='R']
 7 SET [TURN,RIGHT]
 8 IF [(9),(10),(11)]
 9 TRUE [T='B']
10 SET [TURN,BACK]
11 SET [TURN,STRAIGHT]
12 IF [(13),(14),(24)]
13 TRUE [TURN='LEFT']
14 MOVEOUT ['TURNLEFT',D]
15 IF [(16),(17),(24)]
16 TRUE [TURN='RIGHT']
17 MOVEOUT ['TURNRIGHT',D]
18 IF [(19),(20),(24)]
19 TRUE [TURN='STRAIGHT']
20 MOVEOUT ['FORWARD',D]
21 IF [(22),(23)]
22 TRUE [TURN='BACK']
23 MOVEOUT ['BACKWARD',D]
24 END
```

ENGLISH PROGRAM ID # 6, EXPONENTIAL

Number of Tokens = 100
Number of Propositions = 11
Propositional Density = 11%

```
 1 SET [X,0.5]
 2 SET [S,0.25]
 3 SET [L,0.00001]
 4 REPEAT [(9)]
 5 IF [(6),(7),(8)]
 6 TRUE [EXP(X)<3X]
 7 SET [X,X-S]
 8 SET [X,X+S]
 9 SET [S,S/2.0]
10 TRUE [S<L]
11 MOVEOUT ['X= ',X]
```

ENGLISH PROGRAM ID # 2

Number of Tokens = 92
Number of Propositions = 10
Propositional Density = 11%

```
 1 MOVEIN [S,NTIMES]
 2 REPEAT [I,1,NTIMES]
 3 CALL [R,S]
 4 SET [S,(5)]
 5 MOD [(S*PRIME+1),LARGESTNO]
 6 SET [R,(7)]
 7 VALUE [S/LARGESTNO]
 8 RETURN [3]
 9 MOVEOUT [X]
10 INCREMENT [I]
```

ENGLISH PROGRAM ID # 3, FACTORIAL

Number of Tokens = 78
Number of Propositions = 9
Propositional Density = 12%

```
1 CALL (F,X)
2 IF [(3),(4),(5)]
3 TRUE [N<=1]
4 SET [F,N-1]
5 IF [(6),(7),(8)]
6 TRUE [N>1]
7 SET [F,N*(8)]
8 CALL [F,N-1]
9 RETURN [F]
```

ENGLISH PROGRAM ID # 4, FIBONACCI

Number of Tokens = 50
Number of Propositions = 9
Propositional Density = 18%

```
1 SET [J,0]
2 SET [K,1]
3 MOVEOUT ['K=']
4 REPEAT [(5)]
5 WHILE [K<1000]
6 MOVEOUT [K]
7 SET [K,J+K]
8 SET [J,K-J]
9 END [4]
```

ENGLISH PROGRAM ID #11, FREQUENCY

Number of Tokens = 73
Number of Propositions = 10
Propositional Density = 14%

```
 1 REPEAT [I,1,NITEMS]
 2 SET [F[I],0]
 3 INCREMENT [I]
 4 REPEAT [I,1,NITEMS]
 5 MOVEIN [NUMBER]
 6 IF [(7),(8)]
 7 TRUE [NUMBER <= NITEMS]
 8 INCREMENT [F(NUMBER)]
 9 MOVEOUT ['VALUE IN ',I, 'IS',F(I)]
10 END [4]
```

APPENDIX D. RECOGNITION MEMORY QUESTIONS

PASCAL SET 2

PROGRAM 1D # 9

1 Which of the following statements were in the previous program?

A. IF V[K] < X THEN X := V[K]
B. IF X[K] < V THEN V := X[K]
C. IF V[K] > X THEN X := V[1]
D. IF X[K] > V THEN V := X[K]
E. IF V[K] > X THEN X := V[K]

2 Which of the following statements is true concerning the previous program?

A. The variable V is a character string.
B. V is an array of real numbers.
C. The variable V is the constant 100.
D. V is an array of integers.
E. The variable V is a real number.

3 Which of the following statements gives the purpose of the previous program?

A. It prints the smallest element in the array.
B. It prints the largest element in the array.
C. It prints the average value of the elements in the array.
D. It prints any element of the array that is greater than an initial value.
E. It prints any element of the array that is less than an initial value.

4 Which of the following statements is true concerning the previous program?

A. " parameter error" is written if I < 0.
B. " parameter error" is written if L <= 0.
C. " parameter error" is written if I <= 2.
D. " parameter error" is written if L > 0.
E. " parameter error" is written if I < 2.

5 Which of the following is the appropriate calling sequence for the function in the previous program?

A. V(A, L)
B. A(L, X)
C. V(L, A)
D. A(V, L)
E. V(A, I)

PROGRAM ID #10

1 What is the purpose of the function used in the previous program?

A. To find the mean of the values in X.
B. To find the sum of the odd numbers in X.
C. To find the sum of the values in X.
D. To find the median of X.
E. To find the mode of X.

2 If N is less than one, what happened in the previous function M?

A. The value of the function was made zero.
B. The value of the function was set to one.
C. The value of the function became N.
D. The value of the function was X[1].
E. None of the above.

3 If the value of N is odd, which position(s) of X determines the value of M?

A. N divided by two.
B. N plus one.
C. N divided by two plus one.
D. The quantity of N plus one divided by two.
E. The average of the value at N divided by two and the value at the next position.

4 The previous function is called by which statement?

A. M(X, N)
B. X(M, N-1)
C. N(X, M)
D. X(M, N-1)
E. M(X, N-1)

5 If a value of N is not odd, M becomes the value corresponding to which position of X?

A. [N DIV 2]
B. [N DIV 2 + 1]
C. [N + 1 DIV 2]
D. The average of B. and C.
E. The average of A. and B.

PROGRAM ID #7

1 How was the previous program initialized?

A. R was made equal to S and S was made equal to X.
B. S was made equal to R and X was made equal to 1.
C. X was made equal to S and R was made equal to S.
D. R was made equal to X and S was made equal to 1.
E. X was made equal to R and S was made equal to 1.

2 The loop will continue until which of the conditions from the previous program is met?

A. X - R is greater than 0.000001
B. |R - S| is less than or equal to 1.0
C. X - R is less than or equal to 0.000001
D. |S - R| is greater than 0.000001
E. |S - R| is less than or equal to 0.000001

3 The purpose of the previous program is which of the following?

A. to compute the difference of R and S.
B. use a series to compute the Sine of X.
C. use a series to compute the square root of X.
D. use a series to compute e to the X power.
E. None of the above.

4 Which of the following was used to call the function?

A. Q(X)
B. R(X)
C. X(S)
D. S(X)
E. R(S)

5 Which equation was used to compute the next value of S in the previous program?

A. (R * R + S) / (2.0 * R)
B. (X * X + S) / (2.0 * S)
C. (R * R + X) / (2.0 * R)
D. (R * R + X) / (2.0 * S)
E. (R * R + S) / (2.0 * X)

PROGRAM ID # 1

1 The purpose of procedure in the previous program is which of the following?

A. to find the difference between k and j.
B. to find which is greater j or k.
C. to find the remainder when k is divided by j.
D. to switch the values of N1 and N2.
E. to switch the values of N2 and N3.

2 Which statement was used to call the procedure in the previous program?

A. V(N1,N2,N3)
B. X(N3,N2,N1)
C. N(N3,N2,N1)
D. M(N1,N2,N3)
E. None of the above.

3 The final answer in the previous program was left in which variable?

A. N1
B. N2
C. N3
D. I
E. K

4 What is the purpose of the previous program?

A. To find the smallest of N1, N2, and N3.
B. To find the largest of N1, N2, and N3.
C. To find the largest common denominator of N1 and N2.
D. To find the largest common denominator of N2 and N3.
E. To find a common multiple of N1, N2, and N3.

5 Which condition determines how long to continue the loop?

A. While N1 is not zero.
B. While N2 is not zero.
C. While N3 is not zero.
D. As long as N2 is equal to zero.
E. As long as N3 is equal to zero.

PROGRAM ID # 8

1 Which action will the function in the previous program perform?

A. If M < 0, the value of L becomes zero.
B. If M <= 0, the value of S becomes zero.
C. If L <= 0, the value of M becomes zero.
D. If L > 0, the value of M becomes zero.
E. If I < 0, the value of S becomes zero.

2 The purpose of the previous program is which of the following?

A. To find the average value of an array.
B. To find the maximum value of an array.
C. To find the minimum value of an array.
D. To find the median value of an array.
E. None of the above.

3 The variable in the previous function which held the number of the values was which of the following?

A. N
B. M
C. S
D. X
E. L

4 The function was called in the previous program by which statement?

A. M(V,L)
B. V(L,N)
C. V(X,L-1)
D. M(V,L-1)
E. V(X,N-1)

5 Which equation was used in the previous function?

A. M := S / L
B. L := S / M
C. M := X / L
D. L := X / S
E. M := S / N

PROGRAM ID #12

1 Which of the following statements appear in the previous program?

```
A. WHILE (FLAG) OR (N > FACTOR * FACTOR)
B. WHILE (FLAG) AND (N > FACTOR * FACTOR)
C. WHILE (FLAG) OR ( FACTOR * FACTOR > N)
D. WHILE (FLAG) AND (FACTOR * FACTOR > N)
E. WHILE (FLAG) OR (N < FACTOR * FACTOR)
```

2 Which statement did you see in the previous program?

```
A. IF N NOT ODD THEN P:=FALSE
B. IF N EVEN THEN P := FALSE
C. IF NOT ODD(N) THEN P := TRUE
D. IF NOT EVEN(N) THEN P := TRUE
E. IF NOT ODD(N) THEN P := FALSE
```

3 Indicate which program statement was previously printed.

```
A. IF FACTOR MOD N <> 0 THEN FACTOR := FACTOR + 1
B. IF FACTOR MOD N = 0 THEN FACTOR := FACTOR + 2
C. IF N MOD FACTOR <> 0 THEN FLAG = FALSE
D. IF N MOD FACTOR = 0 THEN FACTOR = FACTOR + 1
E. IF N MOD FACTOR <> 0 THEN FACTOR = FACTOR + 2
```

4 What program statement previously appeared?

```
A. IF N  > 1 THEN P := TRUE
B. IF N <= 1 THEN P := TRUE
C. IF N <= 1 THEN P := FALSE
D. IF N <= 1 THEN P := FALSE
E. IF N <  1 THEN P := FLAG
```

5 Which of the following statements best describes the purpose of the previous program?

A. It prints the first MX odd numbers.
B. It prints the first MX even numbers.
C. It prints MX random numbers.
D. It prints the first MX prime numbers.
E. It prints the first MX numbers of a Fibonacci series.

PASCAL SET 1

PROGRAM ID # 5

1 Which of the following statements appeared in the previous algorithm?

A. IF T = 'B' THEN TURN := BACKWARD
B. IF T = 'S' THEN TURN := STRAIGHT
C. IF T = 'R' THEN TURN := ROTATE
D. IF T = 'F' THEN TURN := FORWARD
E. IF T = 'B' THEN TURN := BACK

2 Which of the values was not given to Turn in the previous algorithm?

A. LEFT
B. RIGHT
C. FORWARD
D. ST
E. BACK

3 The purpose of the preceding algorithm is which of the following?

A. to establish a quantity for the direction
B. to establish which direction to turn
C. to find out how far to rotate an angle
D. both A and C
E. both A and B

4 Which of the followng was not written in the previous algorithm?

A. 'TURN RIGHT'
B. 'TURN LEFT'
C. 'TURN BACK'
D. 'FORWARD'
E. all of these messages were written

5 Which of the following statements best explain D in the previous algorithm?

A. a letter that gives the type of direction
B. the letter for "down" should be put into Turn
C. a number which specifies a quantity of Turn
D. a letter that is put into T
E. none of the above

PROGRAM ID # 6

1 In the previous algorithm, what was the initial value of X?

A. 1.0
B. 1.0E-6
C. 0.5
D. 0.25
E. 1.0E-3

2 What was the value of L in the previous algorithm?

A. 1.0
B. 1.0E-6
C. 0.5
D. 0.25
E. 1.0E-3

3 In the previous algorithm, what condition determined the termination of the loop?

A. UNTIL S >= L
B. UNTIL S < L
C. UNTIL S > X
D. UNTIL X < L
E. UNTIL EXP(X) < L

4 The purpose of the previous algorithm is to find which of the following?

A. the square root of X
B. X, where e raised to the power X is equal to three times X
C. the value of e raised to power X
D. the square root of e raised to the power X
E. none of the above

5 In the preceeding algorithm, which statement is the condition which determines X := X - S

A. when X < L
B. when EXP(X) < 3 * X
C. when EXP(X) > 3 * X
D. when EXP(X) = 3 * X
E. when C or D is true

PROGRAM ID # 2

1 Which of the following is true for the previous algorithm?

A. it uses the function S to calculate a value of R
B. it uses the function R to calculate a value of S
C. it multiplies R by a constant
D. it divides R by the computer's largest integer
E. none of the above

2 The purpose of the previous algorithm is which of the following?

A. to find the remainder of S
B. to generate an arithmetic series of S
C. to calculate a random number
D. to calculate the square root of S
E. none of the above

3 Not including the end-points, the value of the answer in the previous algorithm should be in what range ?

A. 1 TO 65536
B. -65536 TO 0
C. 0 TO 65536
D. 0 TO 1
E. none of the above

4 An appropriate constant in the previous algorithm would be which of following?

A. 0
B. 1
C. 2
D. 5913
E. 65536

5 In the previous algorithm, if modulo arithmetic were not used which of the following statements indicate what would happen after the function was used several times?

A. the answer would become zero
B. the answer would remain the same
C. the answer would exceed the computer's word size
D. the answer would become one
E. none of the above

PROGRAM ID # 3

1 In the previous algorithm, which of the following statements is true?

A. Function N is first given the value of F
B. Function F is first given the value of one
C. Function N is first given the value of F minus one
D. Function F is first given the variable N
E. None of the above

2 The final value of the previous algorithm in terms of N is which of the following statements?

A. N * (N - 1) / 2.0
B. N /(N + 1)
C. N - 1
D. N!
E. none of the above

3 Which of the following statements determines the terminal condition of the previous algorithm?

A. when N > 1
B. when N <= 1
C. when N = 0
D. UNTIL I = N * N
E. none of the above

4 In the previous algorithm which statement reflects what happens to N every time the function is used?

A. N is increased by one
B. N is multiplied by N plus one
C. N is decreased by one
D. N is divided by N plus one
E. none of the above

5 If the initial value of N is one, which statement tells what the previous algorithm does?

A. N would become equal to zero
B. the value of the function would become zero
C. the value of the function becomes one
D. one is added to the value of N
E. none of the above

PROGRAM ID #4

1 Which of the following statements is true about the preceding algorithm?

A. J := 0 and K := 1
B. J := 1 and K := 0
C. K := 0 and J := 0
D. K := 1 and J := 1
E. none of the above

2 In the previous algorithm, which of the following events are repeated?

A. J := K + J
B. J := J * K
C. K := K - J
D. J := K - J
E. none of the above

3 How many times is the loop in the previous algorithm repeated?

A. WHILE (K < 1000)
B. WHILE (K < 10000)
C. 10000 times
D. 100000 times
E. none of the above

4 Which of the following is the purpose of the previous algorithm?

A. to calculate a sequence of odd numbers
B. to calculate a sequence of prime numbers
C. to calculate a sequence of random numbers
D. to calculate a type of arithmetic series called a Fibonacci series
E. none of the above

5 Which of the following events occurs in the previous algorithm?

A. the value of J is written
B. the value of K is written
C. the title "J" is written
D. the value of K-J is written
E. none of the above

PROGRAM ID #11

1 Which statement tells how was F used in the previous algorithm?

A. it was a terminal value for the data
B. it was an array to store the values
C. it was used to calulate the sum of the values
D. it is used to store the frequency of the values
E. none of the above

2 The values of F in the previous algorithm are determined by which of the following statements?

A. F[N] := X
B. F[N] := L
C. F[N] := F[N] + 1
D. A and B
E. none of the above

3 The purpose of the previous algorithm is which of the following?

A. to keep all the values in a certain range
B. to find a maximum value
C. to round all of the values
D. to find the frequency of each of the values
E. none of the above

4 Which values were written in the previous algorithm?

A. all the values that were checked
B. all the values less than or equal to zero
C. all the positive numbers
D. all the frequency counts in F greater than zero
E. none of the above

5 Which of the statements concerning the previous algorithm is true?

A. the values can't be integers
B. the values must be negative
C. the range of values must be within the array
D. all the values must be odd
E. none of the above

ENGLISH RECOGNITION MEMORY TEST

SET 1

PROGRAM ID # 9

1 Which of the following statements were in the previous program?

A. each time an element of V is less than X, it replaces X
B. each time an element of X is less than V, it replaces V
C. each time an element of L is greater than X, it replaces X
D. each time an element of X is greater than L, it replaces L
E. each time an element of V is greater than X, it replaces X

2 Which of the following statements is true concerning the previous program?

A. the value of V becomes the value of function X
B. the value of X becomes the value of function A
C. the value of L becomes the value of function A
D. function A become the value of L minus one
E. none of the above

3 Which of the following statements gives the purpose of the previous program?

A. It prints the smallest element in the array.
B. It prints the largest element in the array.
C. It prints the mean of thevalues in the array.
D. It prints any element of the array that is greater than an initial value.
E. It prints any element of the array that is less than an initial value.

4 Which of the following statements is true concerning the previous program?

A. "parameter error" is written if I is less than zero
B. "parameter error" is written if L is less than or equal to zero
C. "parameter error" is written if I less than or equal to two
D. "parameter error" is written if L greater than zero
E. "parameter error" is written if I less than two

5 Which of the following occurs in the previous program if L equals one?

A. the function gets the value of X
B. the function gets the value of one
C. the function gets the value of zero
D. the function returns an error message
E. none of the above

PROGRAM ID #10

1 What is the purpose of the function used in the previous program?

A. To find the mean of the values in X.
B. To find the sum of the odd numbers in X.
C. To find the sum of the values in X.
D. To find the median of X.
E. To find the mode of X.

2 If N is less than one, what happened in the previous function M?

A. The value of the function was made zero.
B. The value of the function was set to one.
C. The value of the function became N.
D. The value of the function was X[1].
E. None of the above.

3 If the value of N is odd, which position(s) of X determines the value of M?

A. N divided by two.
B. N plus one.
C. N divided by two plus one.
D. The quantity of N plus one divided by two.
E. The average of the value at N divided by two and the value at the next position.

4 The function in the previous algorithm uses which of the following?

A. the function M uses values of X and N
B. the function X uses values of X and N
C. the function N uses values of M and N minus one
D. the function X uses values of M and N minus one
E. the function M uses values of X and N minus one

5 If a value of N is not odd, M becomes the value corresponding to which position of X?

A. N divided by two
B. N divided by the quantity two plus one
C. N plus one divided by two
D. The average of B. and C.
E. The average of A. and B.

PROGRAM ID # 7

1 How was the previous program initialized?

A. R was made equal to S and S was made equal to X.
B. S was made equal to R and X was made equal to 1.
C. X was made equal to S and R was made equal to S.
D. R was made equal to X and S was made equal to 1.
E. X was made equal to R and S was made equal to 1.

2 The loop will continue until which of the conditions from the previous program is met?

A. X minus R is greater than .000001
B. the absolute value of R - S is less than or equal to 1.0
C. X minus R is less than or equal to .000001
D. the absolute value of S - R is greater than .000001
E. the absolute value of S - R is less than or equal to .000001

3 The purpose of the previous program is which of the following?

A. to compute the difference of R and S.
B. use a series to compute the Sine of X.
C. use a series to compute the square root of X.
D. use a series to compute e to the X power.
E. None of the above.

4 Which of the following shows how the variables are used?

A. the function Q uses the value X
B. the function R uses the value X
C. the function X uses the value S
D. the function S uses the value X
E. the function R uses the value S

5 How was the next value of S computed in the previous program?

A. the quantity R squared plus S divided by two times R
B. the quantity X squared plus S divided by two times S
C. the quantity R squared plus X divided by two times R
D. the quantity R squared plus X divided by two times S
E. the quantity R squared plus S divided by two times X

PROGRAM ID # 1

1 The purpose of procedure in the previous program is which of the following?

A. to find the difference between k and j.
B. to find which is greater j or k.
C. to find the remainder when k is divided by j.
D. to switch the values of N1 and N2.
E. to switch the values of N2 and N3.

2 Which statement shows the relationship of the variables in the previous algorithm?

A. an internal procedure V uses N1, N2, and N3
B. an internal procedure X uses N3, N2, and N1
C. an internal procedure N uses N3, N2, and N1
D. an internal procedure M uses N1, N2, and N3
E. None of the above.

3 The final answer in the previous program was left in which variable?

A. N1
B. N2
C. N3
D. I
E. K

4 What is the purpose of the previous program?

A. To find the smallest of N1, N2, and N3.
B. To find the largest of N1, N2, and N3.
C. To find the largest common denominator of N1 and N2.
D. To find the largest common denominator of N2 and N3.
E. To find a common multiple of N1, N2, and N3.

5 Which condition determines how long to continue the loop?

A. While N1 is not zero.
B. While N2 is not zero.
C. While N3 is not zero.
D. As long as N2 is equal to zero.
E. As long as N3 is equal to zero.

PROGRAM ID # 8

1 Which statement shows the action of the function in the previous algorithm perform?

A. If M less than zero, the value of L becomes zero.
B. If M less than or equal to zero, the value of S becomes zero.
C. If L less than or equal to zero, the value of M becomes zero.
D. If L is greater than zero, the value of M becomes zero.
E. If I is less than zero, the value of S becomes zero.

2 The purpose of the previous program is which of the following?

A. To find the average value of an array.
B. To find the maximum value of an array.
C. To find the minimum value of an array.
D. To find the median value of an array.
E. None of the above.

3 The variable in the previous function which held the number of the values was which of the following?

A. N
B. M
C. S
D. X
E. L

4 The function was called in the previous program by which statement?

A. function M uses variables V and L
B. function V uses variables L and N
C. function X uses variables L and N
D. function L uses variables V and X
E. none of the above

5 How was the final value computed in the previous function?

A. S was divided by L
B. L was divided by S
C. X was divided by L
D. X was divided by S
E. M was divided by S

PROGRAM ID #12

1 Which of the following statements is included in previous algorithm?

A. P is true or N is greater than factor squared
B. P is true and N is greater than factor squared
C. P is true or factor squared is greater than N
D. P is true and factor squared is greater than N
E. P is true or N is less than factor squared

2 Which statement is part of the previous algorithm?

A. if N is less than two, P is true
B. if N is equal to two, P is true
C. if N is greater than two and is even, P is true
D. if the remainder of N divided by factor is zero, P is false
E. none of the above

3 Indicate which event occured in the previous algorithm

A. factor is increased by one, if N divides evenly into factor
B. factor is increased by two, if N divides evenly into factor
C. the flag is set to true if N divides evely into factor
D. factor is increased by two, if factor doesn't divide evenly by N
E. none of the avove

4 Which statement sets the flag true in the previous algorithm?

A. N is equal to one
B. N is less or equal to one
C. N is even and greater than two
D. N is equal to three
E. none of the above

5 Which of the following statements best describes the purpose of the previous algorithm?

A. to identify odd or even numbers.
B. to compute random numbers.
C. to compute whether the numbers belong to a Fibonacci series
D. to calculate whether a number is prime
E. none of the above

SET 2

PROGRAM ID # 5

1 Which of the following statements appeared in the previous algorithm?

A. If T is equal to the letter B then Turn becomes backward
B. If T is equal to the letter S then Turn becomes straight
C. If T is equal to the letter R then Turn becomes rotate
D. If T is equal to the letter F then Turn becomes forward
E. If T is equal to the letter B then Turn becomes back

2 Which of the values was not given to Turn in the previous algorithm?

A. Left
B. Right
C. Forward
D. Straight
E. Back

3 The purpose of the preceding algorithm is which of the following?

A. to establish a quantity for the direction
B. to establish which direction to turn
C. to find out how far to rotate an angle
D. both A and C
E. both A and B

4 Which of the followng was not written in the previous algorithm?

A. Turn Right D
B. Turn Left D
C. Turn Back D
D. Forward D
E. all of these messages were written

5 Which of the following statements best explain D in the previous algorithm?

A. a letter that gives the type of direction
B. the letter that specifies down should be put into Turn
C. a number which specifies a quantity of Turn
D. a letter that is put into T
E. none of the above

PROGRAM ID # 6

1 In the previous algorithm, what was the initial value of X?

A. one
B. one hundred thousandth
C. one half
D. one fourth
E. one thousandth

2 What was the value of L in the previous algorithm?

A. one
B. one hundred thousandth
C. one half
D. one fourth
E. one thousandth

3 In the previous algorithm, what condition determined the termination of the loop?

A. when S became greater than L
B. when S became less than or equal to L
C. when S became greater than X
D. when X becomes less than L
E. when e raised to the power X is less than L

4 The purpose of the previous algorithm is to find which of the following?

A. the square root of X
B. X, where e raised to the power X is equal to three times X
C. the value of e raised to power X
D. the square root of e raised to the power X
E. none of the above

5 In the preceeding algorithm, which statement is the condition which determines when S subtracted from X?

A. when X is less than L
B. when e raised to the power X is less than three times X
C. when e raised to the power X is greater than three times X
D. when e raised to the power X is equal to three times X
E. when C or D is true

PROGRAM ID # 2

1 Which of the following is true for the previous algorithm?

A. it uses the function S to calculate a new value of R
B. it uses the function R to calculate a new value of S
C. it multiplies R by a constant
D. it divides R by the largest integer of the computer
E. none of the above

2 The purpose of the previous algorithm is which of the following?

A. to find the remainder of S
B. to generate an arithmetic series of S
C. to calculate a random number
D. to calculate the square root of S
E. none of the above

3 Not including the end-points, the value of the answer in the previous algorithm should be in what range?

A. from one to the largest value in the computer
B. from the smallest value in the computer to zero
C. from zero to the largest value in the computer
D. from zero to one
E. none of the above

4 An appropriate constant in the previous algorithm would be which of following?

A. zero
B. one
C. two
D. a prime number
E. the largest value in the computer

5 In the previous algorithm, if modulo arithmetic were not used which of the following statements indicate what would happen after the function was used several times?

A. the answer would become zero
B. the answer would remain the same
C. the answer would exceed the computer's word size
D. the answer would become one
E. none of the above

PROGRAM ID # 3

1 In the previous algorithm, which of the following statements is true?

A. Function N is originally given the value of F
B. Function F is originally given the value of one
C. Function N is originally given the value of F minus one
D. Function F is originally given the variable N
E. None of the above

2 The final value of the previous algorithm in terms of N is which of the following statements?

A. N times the quantity N minus one divided by two
B. N divided by the quantity N plus one
C. N minus one
D. N factorial
E. none of the above

3 Which of the following statements determines the terminal condition of the previous algorithm?

A. when N is greater than one
B. when N is less than or equal to one
C. when N becomes zero
D. after repeating the function N squared times
E. none of the above

4 In the previous algorithm which statement reflects what happens to N every time the function is used?

A. N is increased by one
B. N is multiplied by N plus one
C. N is decreased by one
D. N is divided by N plus one
E. none of the above

5 If the initial value of N is one, which statement tells what the previous algorithm does?

A. N would become equal to zero
B. the value of the function would become zero
C. the value of the function becomes one
D. one is added to the value of N
E. none of the above

PROGRAM ID # 4

1 Which of the following statements is true about the preceding algorithm?

A. J is set equal to zero and K is set equal to one
B. J is set equal to one and K is set equal to zero
C. Both K and J are set equal to zero
D. Both K and J are set equal to one
E. none of the above

2 In the previous algorithm, which of the following events are repeated?

A. J becomes K plus J
B. J becomes J times K
C. K becomes K minus J
D. J becomes K minus J
E. none of the above

3 How many times is the loop in the previous algorithm repeated?

A. as long as K is less than one thousand
B. as long as K is less than ten thousand
C. ten thousand times
D. one hundred thousand times
E. none of the above

4 Which of the following is the purpose of the previous algorithm?

A. to calculate a sequence of odd numbers
B. to calculate a sequence of prime numbers
C. to calculate a sequence of random numbers
D. to calculate a type of series called Fibonacci
E. none of the above

5 Which of the following events occurs in the previous algorithm?

A. the value of J is written
B. the value of K is written
C. the title "J" is written
D. the value of K-J is written
E. none of the above

PROGRAM ID #11

1 Which statement tells how was F used in the previous algorithm?

A. it was a terminal value for the data
B. it was an array to store the values
C. it was used to calulate the sum of the values
D. it is used to store the frequency of the values
E. none of the above

2 The values of F in the previous algorithm are determined by which of the following statements?

A. the value of the numbers
B. the total number of values
C. the frequency of the values
D. A and B
E. none of the above

3 The purpose of the previous algorithm is which of the following?

A. to keep all the values in a certain range
B. to find a maximum value
C. to round all of the values
D. to find the frequency of each of the values
E. none of the above

4 Which values were written in the previous algorithm?

A. all the values that were checked
B. all the values less than or equal to zero
C. all the positive numbers
D. all the frequency counts in F greater than zero
E. none of the above

5 Which of the statements concerning the previous algorithm is true?

A. the values can't be integers
B. the values must be negative
C. the range of values must be within the array
D. all the values must be odd
E. none of the above

APPENDIX E. CLOZE TESTS

SET ONE

ID # 9

This program uses values in an _____ called V. The number of these ______ are put into a ________ L. The function A uses the _____ of V and L. _____ are three conditions which ______ on the value of L. If _ is less than one, a _______"parameter error", is written. Otherwise, the _____ element of V is put into a _________ variable called X. If L is _____ to one, the value of X _______ the answer. If L is _______ than one, successive comparisons are ____. Beginning with the second _______, every element of V is ________ and compared to the current _____ contained in X. Each time the _______ is greater than X, it ________ the current value of _. After all the comparisons have ____ made, the value that is _________ in X becomes the answer.

ID #10

This program uses a function _____ M. M is given a set of ordered ______ which have been placed in an _____ called X. The number of the values is ___ into a variable called, N. If N is ____ than one, the value of the ________ M becomes zero. Otherwise, if _ is odd then M _______ the value which corresponds to the ________ number obtained by dividing ___ into N and adding ___ If N is not odd, then _ becomes the value obtained by ________ two into the sum of the ______ that correspond to the position N divided by ___ and the position _ divided by two plus one.

ID # 7

A function Q uses a predefined ________ X. The variable, R, is made _____ to X. The initial value of a variable, S, is ____ equal to one. A ____ is made to see if the ________ value of S minus _ is within a 0.000001 tolerence. If it isn't, the _________ actions are repeated: first _ is made equal to the ______ value of S, then R is _______ and added to X, next this ___ is divided by two times R, this ________ then becomes the new value of _. When S is within the required .000001 _____ of the value of R, the _________ ends with the value of the procedure ________ from the variable, S.

ID # 1

In a program that has _____ N2 and N3 initially set, the following _____ happen as long as there is ____: The value of N1 is made equal to the _____ of N2 and the value of N2 becomes __. An internal procedure M, ____ N1, N2, and N3 by the _____ of I, J, and K, as follows: _____ K is greater than or _____ to J, J is subtracted from _ and that difference becomes the ____ value of K. As soon as the _____ of K is less ____ the value of J the _____ ends with the result ____ in the variable, _. This procedure is used _____ N3 equals zero. Then the _____ N2 contains the answer.

ID # 8

There is function, M, which ____ a set of values V, and L, the _____ of the values in the set. S is _____ equal to zero. If L is ____ than or equal to zero, an _____ is indicated and the function, M is ___ to zero Otherwise no error _____ is given, each element of the ___ is added to the accumulative sum of S, and the _____ of S, the answer of procedure M, is _____ by dividing L into S.

ID #12

In order to determine if a function P has a ____ or false value, the following tests are _____: If N is less than two, the _____ of P is false. If N is _____ to two, the value of P is ____. However, if N is any other even number, the _____ of P is false. For any other N the _____ steps are taken: A variable _____ Factor is set equal to three and P is initially _____ a value of true. As long as the value of P is ____ and N is greater than _____ squared, the following test is _____ If the remainder of N divided by Factor is not _____ to zero, the value of Factor is _____ by two. If the remainder is equal to ____ then the value of P becomes _____. The value of P will remain ____ only when the remainder of _ divided by Factor doesn't become ____ during the test.

ID # 5

```
PROGRAM THREE (INPUT,OUTPUT);
   TYPE DIRECTION = (LEFT,_____,BACK,STRAIGHT);
   VAR TURN _ DIRECTION;
          D : INTEGER;
          _ : CHAR;
   BEGIN
     READLN(T,D);
     __ T = 'L' THEN ____ := LEFT
       ELSE IF _ = 'R' THEN TURN __ RIGHT
       ELSE IF T _ 'B' THEN TURN := ____
       ELSE TURN := STRAIGHT;
     ____ TURN OF
       LEFT  : _______ ( 'TURN LEFT',D);
       RIGHT _ WRITELN ( 'TURN RIGHT',D);
       ________ : WRITELN ( 'FORWARD',D);
       ____ : WRITELN ( 'BACKWARD',D)
     ___
  END.
```

ID # 6

```
PROGRAM SIX(OUTPUT);
   VAR X,S,L _ REAL;
   BEGIN
     X := ___;
     S := 0.25;
     L __ 1.0E-6;
   REPEAT
     IF EXP(_) < 3 * X
        ____ X := X - _
       ELSE
         BEGIN
           X := _ + S;
           S := _ / 2.0
         END
   UNTIL _ < L;
   WRITELN ('X =', _:8:6)
  ___.
```

ID # 2

```
PROGRAM EIGHT (INPUT,OUTPUT);
   CONST N = 100;
   ___   S, I : INTEGER;
   ________ R (VAR S:_______) : REAL;
    CONST A _ 5913;
          C = 1;
          _ = 65536;
    BEGIN
     S __ (A * S + _) MOD M;
     R := _ / M
    END;
  BEGIN
   ______ (S);
   FOR I := _ TO N DO
     WRITELN (' _=',R(S))
  END.
```

ID # 3

```
PROGRAM NINE (INPUT,OUTPUT);
   VAR X,N : _______;
   FUNCTION F(N:_______) : INTEGER;
     BEGIN
       IF _ = 1
          THEN F __ N
          ELSE F := _ * F(N - 1)
     END;
  _____
   WHILE NOT EOF DO
     _____
      READLN (X);
      WRITELN ('F=',_(X))
     END
  END.
```

ID # 4

```
PROGRAM TEN (INPUT,OUTPUT);
   VAR K,J : ________;
   BEGIN
     J := 0;
     _ := 1;
     WRITELN ('K=');
     _____ (K < 1000) DO
     _____
       WRITELN (K);
       K := _ + K;
       J := _ - J
     END
 END.
```

ID #11

```
PROGRAM ELEVEN(INPUT,OUTPUT);
   CONST MN = 255;
   ____ LIST = ARRAY [0..MN] __ INTEGER;
   VAR L,N,_ : INTEGER;
           F : LIST;
   _____
     FOR L := 0 __ MN DO
       F[L] __  0;
   READ (T);
   REPEAT
     __ N <> T THEN
        __  (N >= 0) ___ (N <= MN)
           ____ F[N] := F[_] + 1
   UNTIL N _ T;
   FOR L := _ TO MN DO
     BEGIN
       __ F[L] <> 0 ____ WRITELN(' VALUE IN ',L,' IS',_[L])
     END
 END.
```

SET TWO

ID # 5

The value of the variable Turn is _____ in this program from the first letter of the a _____ called T. Also a number D is _____, which specifies the quantity of Turn. If _ is the letter L, then Turn becomes ____. If T is the letter R then ____ becomes Right. If T is _____ to the letter B then Turn becomes ____. If T has any other value Turn becomes Straight. In the ____ where Turn is equal to Left, the _____ "Turn Left" D is written. If Turn _____ Right, the message is "Turn Right" D. If Turn _____ Back, the message is "Backward" D. However if Turn equals Straight, the _____ written is "Forward" D.

ID # 6

The variable X is given the _____ of one half, the variable S is given the value of one _____ and the variable L is given the _____ one hundred thousanth. The following events are _____ until the value of _ is less than the value of L. If e _____ to the power X is less than three times X, X becomes X minus _ Otherwise the value of X becomes the ___ of X plus S and S is _____ in half. When S becomes less than or equal to _, the current value of X is written.

ID # 2

There is a program that initially _____ a value S to a function R. The _____ R multiplies S by a prime _____ and adds the product to one. This ___ then becomes the new value of _. S is then divided by the largest integer, and the _____ becomes the new value of S. S is _____ by the largest integer value again and the _____ becomes the answer to the _____, R. This function is repeatedly used to get new values of _ within the range of zero to one.

ID # 3

Function F is originally given a _____ N. When N is less than or equal to ___, the process ends and the value of _ is returned. Each time F is used, _ is made equal to N minus ___. When N is greater than one, the _____ returned from function _ will be the product of N times the result of using _____ F with the new value of N _____ one assigned as its given variable.

ID # 4

J is initially made equal to zero and _ is made equal to one. A title, "K", is _____. As long as K is less than ___ thousand the following events are repeated: the value of _ is written, K becomes K ____ J, and J becomes K minus J.

ID #11

There is a program which sets an _____ of numbers, F, equal to zero. There are several _____ which are checked to see if they are within the _____ of F, if a number is within the range, ___ is added to the position of the number in the array _. As soon as all the data is checked, all the _____ in the array F which are greater than ____ are written.

ID # 9

```
PROGRAM ONE (INPUT,OUTPUT);
   CONST ML= 100;
   ____ LIST = ARRAY [1..ML] __ REAL;
   VAR V    : LIST; _,L : INTEGER;
   FUNCTION A(_:LIST; L:INTEGER) :REAL;
     ___ I : INTEGER; X : REAL;
       _____
         IF L > 0 ____ BEGIN
                  X := V[1];
                  ___ I := 2 TO _ DO
                    IF V[I] > _ THEN X := V[I]
                ___
             ELSE WRITELN(' PARAMETER ERROR');
        A __ X
        END;
  BEGIN
   L __ 1;
   WHILE ___ EOF DO
     BEGIN
       READLN (____); IF NOT EOF THEN _ := L + 1;
     ___;
   WRITELN('THE VALUE OF X IS EQUAL TO ',A(V,L));
  END.
```

ID #10

```
PROGRAM TWO(INPUT,OUTPUT);
   CONST ML = 100;
   ____ LIST = ARRAY[1..ML] OF ____;
   VAR X : LIST;
       _ : INTEGER;
   FUNCTION M( X:____; N:INTEGER) : ____;
    BEGIN
     IF N < _ THEN M := 0
                ____ IF ODD (N)
                       ____ M := X[(N+1) ___ 2]
              ELSE M := (X[_ DIV 2] + X[_ DIV 2 + 1]) _ 2.0
    END;
  BEGIN
    N __ 1;
    REPEAT
      READ (X[_]);
      N := N + _;
    UNTIL EOF;
    WRITELN ('M=',_(X,N-1))
  END.
```

ID # 7

```
PROGRAM FOUR (INPUT,OUTPUT);
   VAR X : REAL;
   ________ Q(X:REAL) : ____;
     VAR R,S : ____;
     BEGIN
       R := X;
       _ := 1;
       WHILE ABS (_ - R) > 0.000001 DO
         _____
           R := S;
           S __ (R * R + _) / (2.0 * R)
     ___;
     Q := S
   END;
  _____
   WHILE NOT EOF DO
     _____
       READLN(X);
       WRITELN(X,_(X))
     END
  END.
```

ID # 1

```
PROGRAM FIVE(INPUT,OUTPUT);
   VAR N1,N2,N3 _ INTEGER;
   PROCEDURE M( I,_ :INTEGER; VAR K_INTEGER);
     BEGIN
       K := _;
       WHILE ( K >= J) __
         K := K - J
     ___;
  BEGIN
   WHILE NOT EOF __
     BEGIN
       READLN (N2,N3);
       _____ (N2,N3);
       WHILE N3 __ 0 DO
         BEGIN
           N1 __ N2;
           N2 := N3;
           _(N1,N2,N3)
         END;
       _____ (' M IS', N2)
     END
  END.
```

ID # 8

```
PROGRAM SEVEN (INPUT,OUTPUT);
   CONST ML = 100;
   ____ LIST = ARRAY [ 1..ML] __ REAL;
   VAR V : ____; L : INTEGER;
   FUNCTION _( X:LIST; L_INTEGER) : REAL;
     VAR _ : INTEGER; S : ____;
    BEGIN
      IF L > _ THEN BEGIN
           S := _;
           FOR I := 1 __ L DO
              S := _ + X[I];
           M := _ / L
         END
       ELSE _ := 0
    END;
   BEGIN
    _ := 1;
    WHILE NOT ___ DO
     BEGIN
       READ (V[_]); L := L + _
     END;
   WRITELN ( 'M=', M(_,L-1))
  END.
```

ID #12

```
PROGRAM TWELVE (INPUT,OUTPUT);
   VAR MX, X : INTEGER;
   ________ P (N : INTEGER) _ BOOLEAN;
    VAR FACTOR : ________; FLAG : BOOLEAN;
   BEGIN
   __ N <= 1 ____ P := FALSE
     ELSE  __ N = 2 THEN _ := TRUE
     ELSE IF ___ ODD(N) THEN P __ FALSE
     ELSE BEGIN
            FLAG __ TRUE;
            FACTOR := 3;
            _____ (FLAG) AND (N > ______ * FACTOR) DO
            IF _ MOD FACTOR <> 0 ____ FACTOR := FACTOR + _
                                 ELSE FLAG := FALSE;
            _ := FLAG
          END
   END;
 _____
   READLN (MX);
   WRITELN (' THE ____ VALUES FROM 1 TO ',__,' ARE:');
   FOR X := _ TO MX DO
     IF _(X) THEN WRITE(X)
 ___.
```

APPENDIX F. SUPPLEMENTARY TABLES FOR EXPERIMENT TWO

Table 20

Features of Text by Treatment

	NO. LINES	NO. TOKENS	TEXTUAL DENSITY	NO. PROP.	PROP. DENS.	HIER-ARCHY	READ. INDEX
Program							
1 Greatest Common Divisor							
Pascal	22	68	1	24	35	3	80
English	9	122	14	17	14	3	68
2 Random Number Generator							
Pascal	16	60	4	13	22	2	72
English	8	92	12	10	11	2	74
3 Factorial							
Pascal	15	48	3	11	23	3	83
English	6	78	13	9	12	3	82
4 Fibonacci Sequence							
Pascal	13	38	3	9	24	1	81
English	4	50	13	9	18	1	81
5 Direction							
Pascal	18	75	4	26	35	1	88
English	10	119	12	22	19	1	77
6 Exponential Equation							
Pascal	14	54	4	11	20	2	65
English	7	100	14	11	11	2	74

		NO. LINES	NO. TOKENS	TEXTUAL DENSITY	NO. PROP.	PROP. DENS.	HIER-ARCHY	READ. INDEX
7	Square Root							
	Pascal	21	65	3	14	22	2	77
	English	9	119	13	11	9	2	56
8	Mean							
	Pascal	23	93	4	19	20	3	61
	English	7	84	12	14	17	3	74
9	Maximum							
	Pascal	23	104	5	24	23	4	72
	English	12	142	12	18	13	4	75
10	Median							
	Pascal	20	89	5	22	25	2	73
	English	9	126	14	10	8	2	75
11	Frequency							
	Pascal	19	94	5	19	20	2	79
	English	6	73	12	10	14	2	90
12	Prime Numbers							
	Pascal	23	111	4	30	27	4	72
	English	12	162	14	22	14	4	74

Individual Program Means by Depth and Dependent Variable

	Depth = Low			
	Recognition Memory		Cloze	
Program	English	Pascal	English	Pascal
10	80.00	54.58	70.24	83.52
6	71.34	73.01	78.64	81.41
4	71.34	73.01	87.83	90.48
5	66.96	53.04	83.56	91.59
1	64.35	54.78	87.63	81.82
2	50.43	35.65	73.41	76.03
7	48.70	48.70	55.63	88.78
	Depth = high			
	Recognition Memory		Cloze	
Program	English	Pascal	English	Pascal
8	71.30	63.48	77.27	72.72
9	59.13	60.87	74.48	76.99
3	48.70	59.13	81.80	88.07
12	46.96	53.91	76.32	80.44
1	40.87	50.43	71.95	79.72

Depth = Low

	Seconds to Read		Seconds to Answer	
Program	English	Pascal	English	Pascal
10	124.58	149.63	23.85	22.91
6	175.77	126.63	15.17	25.01
4	109.53	102.41	22.37	18.56
5	99.43	95.10	25.79	18.93
11	69.90	191.27	21.24	20.12
2	108.62	130.77	18.87	27.51
7	132.86	197.61	25.01	17.96

Depth = High

	Seconds to Read		Seconds to Answer	
Program	English	Pascal	English	Pascal
8	109.59	228.26	20.36	20.67
9	134.77	171.87	24.83	23.99
3	115.74	104.95	23.56	24.03
12	158.04	254.02	22.78	31.27
1	109.45	213.39	15.20	28.99

Table 22

Influence of Gist-Condition on Recognition Memory

Gist	No-gist		E-gist		P-gist		Both-gist	
Language	E	P	E	P	E	P	E	P
Program								
1	43.1	44.6	30.0	30.0	40.0	73.3	40.0	40.0
2	42.2	33.3	73.3	26.7	-	-	90.0	70.0
3	33.3	46.7	60.0	30.0	36.0	68.0	62.5	77.5
4	63.3	68.3	80.0	65.0	60.0	80.0	95.0	90.0
5	46.7	20.0	74.3	51.4	40.0	60.0	76.0	62.0
6	40.0	46.3	-	-	26.7	60.0	84.7	80.0
7	40.0	46.3	80.0	33.3	53.3	73.3	80.0	60.0
8	33.3	40.0	95.0	60.0	50.0	80.0	82.3	74.6
9	30.0	40.0	77.1	42.9	40.0	85.0	80.0	86.7
10	20.0	40.0	84.0	36.0	70.0	70.0	84.0	60.0
11	35.0	30.0	75.0	35.0	60.0	86.7	90.0	77.5
12	37.5	45.0	54.3	54.3	33.3	40.0	60.0	76.0
Mean	41.6	43.4	72.4	44.4	44.3	70.8	80.2	72.8

Table 23

Number of Recognition Memory Scores in each Gist Condition

Gist treatment	No-gist	E-gist	P-gist	Both-gist
1	13	2	6	2
2	18	3	0	2
3	6	4	5	8
4	12	4	3	4
5	3	7	3	10
6	3	0	3	17
7	16	3	3	1
8	6	4	2	11
9	6	7	4	6
10	1	5	2	15
11	8	4	3	8
12	8	7	3	5
Total	100	50	37	89

Table 24

Mean Program Comprehension Scores by Session and Language

	Recognition Memory Scores (RMEM)			
	English		Pascal	
	Session		Session	
	I	II	I	II
Program				
1	38.18	43.33	50.00	50.91
2	48.33	52.73	30.91	40.00
3	48.33	49.09	58.18	60.00
4	63.33	80.00	72.73	83.64
5	65.00	69.09	52.73	53.33
6	70.00	72.73	83.64	63.33
7	50.91	46.67	48.33	49.09
8	74.55	68.33	63.33	63.64
9	56.36	61.67	61.67	60.00
10	78.18	81.67	53.33	56.36
11	60.00	52.73	69.09	56.67
12	50.91	43.33	68.33	38.18
Average	58.70	61.30	57.79	55.91
N	138	138	138	138

Seconds taken to Answer (SAC)

	English Session		Pascal Session	
	I	II	I	II
Program				
1	20.08	10.72	40.84	16.06
2	21.73	15.75	29.76	25.45
3	28.97	17.66	31.38	17.29
4	28.92	15.22	20.84	16.47
5	31.24	19.85	25.00	13.37
6	27.82	15.68	18.05	12.53
7	34.18	16.60	21.79	13.79
8	28.71	12.70	24.14	16.88
9	35.84	14.73	27.10	20.60
10	31.32	17.01	24.35	21.34
11	24.94	17.20	24.99	15.65
12	26.84	14.06	37.91	24.02
Average	28.33	15.98	27.27	17.74
N	138	138	138	138

Mean Number of Seconds to Read (SREAD)

	English Session		Pascal Session	
	I	II	I	II
Program				
1	133.95	89.03	248.21	175.41
2	113.22	103.59	167.37	97.21
3	120.97	110.03	117.74	93.22
4	121.78	96.17	117.74	88.36
5	109.49	88.44	129.70	63.38
6	234.24	111.98	187.47	75.92
7	172.86	96.20	234.06	157.85
8	157.76	69.44	281.83	169.81
9	184.21	89.44	180.20	162.79
10	172.76	80.32	158.54	139.91
11	79.18	59.77	231.60	154.30
12	195.85	123.38	331.48	169.53
Average	148.88	93.07	200.67	127.51
N	136	138	137	138

Mean Cloze Scores

	ENGLISH		PASCAL	
	Session		Session	
	I	II	I	II
Program				
1	65.01	78.89	79.55	79.88
2	73.00	73.75	73.75	80.17
3	84.85	79.59	85.23	90.91
4	89.09	86.67	85.00	95.45
5	83.08	83.95	85.23	98.18
6	80.00	77.27	74.38	88.43
7	50.63	60.63	90.15	87.53
8	73.64	80.91	80.91	71.80
9	69.46	80.00	78.61	75.50
10	59.66	81.87	82.39	84.56
11	88.58	86.77	75.00	88.64
12	70.82	81.82	81.40	79.55
Average	73.94	79.52	80.25	84.82
N	126	136	132	137

Table 25

Mean Cloze Scores for Each Presentation Order

	Language			
	English		Pascal	
Program	E-First	P-First	E-First	P-First
1	76.39	70.84	80.00	79.65
2	65.00	75.28	67.27	78.61
3	83.33	81.46	87.50	88.24
4	92.00	86.67	90.00	90.62
5	78.85	84.60	94.66	90.74
6	80.00	73.24	90.91	78.61
7	53.13	56.25	96.67	86.59
8	72.00	78.82	74.45	72.25
9	79.54	73.29	80.00	76.15
10	73.44	69.49	85.00	83.11
11	89.28	87.24	87.50	80.15
12	64.21	79.88	82.73	79.80
Average	75.01	77.01	84.72	81.97
N	52	206	60	210

Table 26

Influence of Native Language on Program Comprehension

	RECOGNITION MEMORY SCORES (RMEM)			
	Language			
	English		Pascal	
	Condition		Condition	
	Native	Non-Native	Native	Non-Native
Program				
1	40.00	44.00	50.00	52.00
2	51.11	48.00	34.44	40.00
3	48.89	48.00	34.44	40.00
4	70.00	76.00	76.67	60.00
5	66.67	68.00	54.44	48.00
6	64.44	96.00	72.22	76.00
7	47.78	57.00	50.00	44.00
8	73.33	64.00	61.11	72.00
9	58.89	60.00	61.11	60.00
10	81.11	76.00	53.33	60.00
11	64.44	64.00	58.89	40.00
12	48.89	40.00	50.00	68.00
Average	59.63	61.33	56.67	57.00
N	216	60	216	60

Seconds to Correctly answer (SAC)

	Language			
	English		Pascal	
	Condition		Condition	
	Native	Non-Native	Native	Non-Native
Program				
1	13.77	20.00	27.96	32.71
2	16.07	28.96	20.40	53.10
3	19.09	39.66	17.77	46.55
4	19.40	33.06	19.05	16.79
5	25.17	28.05	17.37	24.56
6	21.17	25.06	13.99	19.43
7	24.61	26.45	17.78	18.62
8	20.05	21.46	20.21	22.31
9	16.99	53.05	20.44	36.78
10	23.83	23.92	20.85	30.32
11	18.39	31.49	18.06	27.54
12	23.83	19.01	28.30	41.94
Average	20.20	29.21	20.18	30.89
N	216	60	216	60

Mean Seconds Taken to Read (SREAD)

	Language			
	English		Pascal	
	Condition		Condition	
	Native	Non-Native	Native	Non-Native
Program				
1	102.17	134.20	210.17	225.00
2	84.47	195.56	124.30	154.04
3	96.84	183.76	104.17	107.76
4	105.27	124.88	100.17	110.49
5	99.02	100.88	91.40	108.42
6	171.24	192.06	124.97	132.26
7	122.72	169.39	213.79	139.39
8	106.71	121.43	213.67	280.76
9	124.93	170.17	148.25	256.90
10	119.71	142.12	150.55	146.33
11	59.66	106.76	179.51	233.57
12	157.29	160.74	215.33	393.32
Average	112.53	150.16	156.50	190.68
N	214	60	215	60

Mean Cloze Scores

	Language			
	English		Pascal	
	Condition		Condition	
	Native	Non-Native	Native	Non-Native
Program				
1	75.56	61.13	81.02	75.07
2	79.71	52.00	76.26	75.00
3	83.01	77.69	86.11	96.88
4	88.89	84.00	89.22	95.83
5	91.40	56.88	95.55	77.33
6	82.94	64.00	79.29	90.91
7	63.75	31.25	90.28	83.40
8	81.76	62.00	76.54	58.98
9	80.97	53.73	80.72	63.55
10	76.17	51.25	85.56	76.20
11	92.44	67.21	81.94	81.25
12	77.09	73.71	82.57	72.74
Average	81.44	61.14	83.73	78.11
N	199	59	215	55

Table 27

Influence of Program Production Ability on Comprehension

	Recognition Memory Scores (RMEM)			
	Language			
	English		Pascal	
	Condition		Condition	
	Below Mean	Above Mean	Below Mean	Above Mean
Program				
1	40.00	41.33	40.00	56.00
2	52.50	49.33	37.50	34.67
3	40.00	53.33	57.50	60.00
4	77.50	68.00	45.00	78.67
5	45.00	78.67	45.00	57.33
6	82.50	65.33	72.50	73.33
7	42.50	52.00	45.00	50.67
8	70.00	72.00	55.00	68.00
9	57.50	60.00	57.50	62.27
10	67.50	86.67	47.50	58.67
11	57.50	68.00	37.50	64.00
12	45.00	48.00	42.50	60.00
Average	56.46	61.89	50.00	60.33
N	96	180	96	180

Mean Seconds to Correctly Answer (SAC)

	Language			
	English		Pascal	
	Condition		Condition	
	Below Mean	Above Mean	Below Mean	Above Mean
Program				
1	19.27	13.02	37.12	24.65
2	22.73	16.81	36.88	22.51
3	26.85	21.80	35.18	18.08
4	26.50	20.16	16.86	19.46
5	37.73	19.43	22.52	17.01
6	31.56	16.93	16.30	14.57
7	23.71	25.70	12.69	20.78
8	23.37	18.76	19.56	21.26
9	37.58	18.03	26.95	22.41
10	24.80	23.34	24.74	21.93
11	17.01	23.50	20.11	20.12
12	21.40	23.52	24.27	35.00
Average	26.04	20.08	24.43	21.48
N	96	180	96	180

Mean Seconds Taken to Read Treatments (SREAD)

	Language			
	English		Pascal	
	Condition		Condition	
	Below Mean	Above Mean	Below Mean	Above Mean
Program				
1	111.21	108.44	228.47	205.35
2	139.60	92.09	127.82	132.34
3	126.22	110.15	105.97	104.40
4	163.87	80.55	95.31	106.20
5	109.27	94.17	98.08	93.51
6	261.89	129.84	114.91	133.32
7	174.18	110.83	165.66	214.65
8	134.43	95.39	176.87	255.66
9	166.86	117.65	173.51	170.99
10	154.92	108.40	140.36	154.58
11	86.26	61.17	212.00	180.21
12	149.58	162.56	262.32	249.60
Average	148.19	105.98	158.44	166.92
N	96	177	96	179

Mean Cloze Scores

	Language			
	English		Pascal	
	Condition		Condition	
	Below Mean	Above Mean	Below Mean	Above Mean
Program				
1	69.46	73.02	77.12	88.11
2	65.71	77.00	72.73	77.58
3	72.95	85.93	91.07	86.67
4	80.00	92.00	88.09	91.67
5	71.40	89.23	86.66	94.22
6	70.00	82.67	83.12	80.61
7	56.25	55.36	85.46	90.56
8	64.29	83.33	61.17	78.89
9	72.80	75.15	65.46	83.14
10	61.46	73.75	78.09	86.42
11	80.95	90.31	83.04	81.25
12	63.93	82.10	76.15	82.73
Average	69.29	80.16	78.76	84.53
N	80	178	91	179

Table 28

Influence of Programming Experience on Program Comprehension

Mean Recognition Memory Scores (RMEM)				
	Language			
	English		Pascal	
	Condition		Condition	
	Below-Mean	Above-Mean	Below-Mean	Above-Mean
Program				
1	40.00	42.22	48.57	53.33
2	41.43	64.44	31.43	42.22
3	41.43	60.00	51.43	71.11
4	70.00	73.33	67.14	82.22
5	61.43	75.56	48.57	60.00
6	72.86	68.89	72.86	73.33
7	45.71	53.33	41.43	60.00
8	62.86	84.44	57.14	73.33
9	55.71	64.44	47.14	66.67
10	77.14	84.44	47.14	66.67
11	54.29	80.00	40.00	77.78
12	38.57	60.00	47.14	64.44
Average	55.12	67.59	50.71	66.11
N	168	108	168	108

Mean Seconds to Correctly Answer (SAC)

	Language			
	English		Pascal	
	Condition		Condition	
	Below-Mean	Above-Mean	Below-Mean	Above-Mean
Program				
1	17.63	11.41	36.34	17.56
2	22.25	13.61	31.97	20.57
3	25.89	19.94	28.52	17.04
4	25.00	18.27	19.41	17.23
5	30.00	17.85	21.49	14.95
6	28.15	12.47	17.30	11.85
7	29.44	18.12	16.80	19.77
8	23.26	15.84	19.38	22.68
9	31.33	14.71	26.75	19.69
10	23.38	24.58	25.53	18.84
11	22.34	19.53	20.23	19.95
12	21.51	24.76	29.25	34.90
Average	25.09	17.59	24.41	19.54
N	168	108	168	108

Mean Seconds Taken to Read (SREAD)

	Language			
	English		Pascal	
	Condition		Condition	
	Below-Mean	Above-Mean	Below-Mean	Above-Mean
Program				
1	107.94	112.09	223.41	270.86
2	120.75	89.74	120.32	197.81
3	126.34	99.25	101.30	147.02
4	131.57	75.86	87.62	125.42
5	106.27	88.77	90.38	102.44
6	206.84	127.43	109.24	157.05
7	142.28	118.22	178.09	227.98
8	115.42	99.39	223.12	236.25
9	140.38	126.04	200.83	126.82
10	116.00	137.93	142.72	160.39
11	80.80	52.93	202.78	173.35
12	160.43	154.32	270.86	227.83
Average	129.58	106.80	162.56	166.16
N	168	106	168	107

Mean Cloze Scores

	Language			
	English		Pascal	
	Condition		Condition	
	Below-Mean	Above-Mean	Below-Mean	Above-Mean
Program				
1	64.66	80.86	74.43	87.96
2	69.62	78,89	71.33	82.83
3	77.75	87.66	89.42	86.11
4	82.86	95.56	87.50	94.44
5	76.91	93.16	88.57	96.30
6	70.00	91.11	79.72	83.84
7	44.89	68.75	86.33	92.59
8	71.54	85.86	64.32	85.80
9	70.87	79.29	71.86	84.97
10	61.46	81.94	77.84	92.36
11	83.12	93.65	79.81	84.72
12	69.24	86.55	75.01	88.89
Average	70.71	85.24	78.71	88.40
N	162	108	162	108

Table 29

Influence of Reading Ability on Program Comprehension

	Mean Recognition Memory Scores (RMEM)			
	Language			
	English		Pascal	
	Reading Condition		Reading Condition	
	Low	High	Low	High
Program				
1	47.50	37.33	35.00	58.67
2	35.00	58.67	27.50	40.00
3	37.50	54.67	45.00	66.67
4	67.50	73.33	60.00	80.00
5	62.50	69.33	42.50	58.67
6	57.50	78.67	57.50	81.33
7	42.50	52.00	32.50	57.33
8	57.50	78.67	52.50	69.33
9	47.50	65.33	50.00	66.67
10	67.50	86.67	42.50	61.33
11	65.00	64.00	42.50	61.33
12	35.00	53.33	50.00	56.00
Average	51.87	64.33	44.79	63.11
N	96	180	96	180

Mean Seconds to Answer (SAC)

	Language			
	English		Pascal	
	Reading Condition		Reading Condition	
	Low	High	Low	High
Treatment				
1	10.94	17.47	13.99	36.99
2	21.59	17.42	23.97	29.40
3	20.90	24.98	21.87	25.18
4	14.84	26.38	14.87	20.53
5	20.84	28.44	16.36	20.30
6	18.23	24.03	10.95	17.42
7	26.42	24.26	13.66	20.26
8	18.20	21.51	16.09	23.11
9	16.86	29.08	19.44	26.42
10	17.55	27.21	19.70	24.62
11	26.79	18.28	16.18	22.22
12	17.45	25.63	28.72	32.63
Average	19.22	23.72	17.98	24.92
N	96	180	96	180

Mean Seconds Taken to Read Treatments (SREAD)

	Language			
	English		Pascal	
	Reading Condition		Reading Condition	
	Low	High	Low	High
Program				
1	92.14	119.34	174.12	234.34
2	112.61	106.49	109.90	141.90
3	115.55	115.84	107.39	103.65
4	87.70	121.17	92.27	107.82
5	96.78	100.84	68.00	109.55
6	146.56	191.35	92.27	146.26
7	103.85	148.33	181.71	206.28
8	92.30	119.47	187.31	250.10
9	91.00	158.11	167.85	174.02
10	75.77	150.61	113.95	168.66
11	81.48	63.72	164.41	205.59
12	114.57	181.23	223.87	270.11
Average	100.86	131.51	140.26	176.67
N	96	178	96	179

Cloze Scores

	Language			
	English		Pascal	
	Reading Condition		Reading Condition	
	Low	High	Low	High
Program				
1	61.13	76.59	70.87	84.44
2	70.63	75.00	66.24	80.61
3	72.17	87.30	89.29	87.50
4	80.00	92.00	75.00	96.67
5	79.78	85.72	84.17	91.11
6	68.57	83.33	71.43	86.06
7	42.71	61.16	84.42	91.11
8	78.57	76.67	65.33	76.67
9	68.90	77.27	72.81	79.22
10	64.29	73.21	77.63	86.67
11	83.67	89.61	77.68	83.75
12	75.95	76.49	75.58	83.03
Average	71.20	79.59	75.88	85.94
N	96	180	90	180

APPENDIX G. PL/I PROGRAMS USED FOR EXPERIMENT TWO

```
/*************************************************************/
/*                                                           */
/*                                                           */
/*************************************************************/

DECLARE EXIT ENTRY EXTERNAL,
        OPEN ENTRY EXTERNAL,
        CLOSE ENTRY EXTERNAL,
        GET ENTRY EXTERNAL,
        PUTCR ENTRY EXTERNAL,
        PUT ENTRY EXTERNAL,
        SMOVE ENTRY EXTERNAL,
        CONCAT ENTRY EXTERNAL,
        WHEX ENTRY EXTERNAL RETURNS(CHARACTER(4) VARYING),
        BHEX ENTRY EXTERNAL RETURNS(CHARACTER(2) VARYING),
        HEXB ENTRY EXTERNAL RETURNS(BINARY FIXED(15)),
        SUBSTR2 ENTRY EXTERNAL,
        SUBSTR3 ENTRY EXTERNAL,
        DEC ENTRY EXTERNAL RETURNS(CHARACTER(6) VARYING),
        BIN ENTRY EXTERNAL RETURNS(BINARY FIXED(15)),
        SCOMP ENTRY EXTERNAL,
        WCOMP ENTRY EXTERNAL,
        WADD ENTRY EXTERNAL,
        WSUB ENTRY EXTERNAL,
        ABS ENTRY EXTERNAL RETURNS(BINARY FIXED(15)),
        NEG ENTRY EXTERNAL,
        MULT ENTRY EXTERNAL,
        DIVIDE ENTRY EXTERNAL,
        MOD ENTRY EXTERNAL RETURNS(BINARY FIXED(15)),
        INDEX ENTRY EXTERNAL RETURNS(BINARY FIXED(15)),
        VERIFY ENTRY EXTERNAL RETURNS(BINARY FIXED(15)),
        CHARIN ENTRY EXTERNAL RETURNS (CHARACTER(1) VARYING),
        WRTM ENTRY EXTERNAL,
        RDTM ENTRY EXTERNAL RETURNS (BINARY FIXED(15)),
        STM ENTRY EXTERNAL,
        GTM ENTRY EXTERNAL;
```

```
DECLARE IN BINARY FIXED (15) EXTERNAL INITIAL(1),
        TWO BINARY FIXED(15) EXTERNAL INITIAL(2),
        FILEIN BINARY FIXED(15) EXTERNAL INITIAL(6),
        CONSOLE BINARY FIXED(15) EXTERNAL INITIAL(7),
        TRUE BIT(1) EXTERNAL INITIAL('1'B),
        FALSE BIT(1) EXTERNAL INITIAL('0'B);
DEXPER: PROCEDURE OPTIONS(MAIN);
%INCLUDE GLOBAL1;

        DECLARE ANSWER CHARACTER(1) VARYING,
                DATA CHARACTER(14) VARYING,
                DEBUG BIT(1) INITIAL('0'B),
                EXTEXT(144) CHARACTER(72) VARYING,
                FILE2 BINARY FIXED(15) INITIAL(8),
                FIRST BINARY FIXED(15),
                HEAD(12) CHARACTER(2) VARYING,
                I BINARY FIXED(15),
                INDX BINARY FIXED(15),
                INITLN(24) BINARY FIXED(15),
                K BINARY FIXED(15),
                LAST BINARY FIXED(15),
                LETTER (2) CHARACTER(1) VARYING,
                LOOP BINARY FIXED(15),
                MXSIZE BINARY FIXED(15) INITIAL(144),
                NAME CHARACTER(10) VARYING,
                NEWMX BINARY FIXED(15),
                NSENT BINARY FIXED(15),
                ORDER(12) CHARACTER(2) VARYING,
                POINT(12) BINARY FIXED(15),
                QMAX BINARY FIXED(15) INITIAL(200),
                QSIZE(72) BINARY FIXED(15),
                QUESTID(24) CHARACTER(2)VARYING,
                RC BINARY FIXED(15),
                RESTLN(200) CHARACTER(72) VARYING,
                SENTNM CHARACTER(14) VARYING,
                SENTMX BINARY FIXED(15),
                SEQUENC BINARY FIXED(15),
                SETNO BINARY FIXED(15),
                SIZE BINARY FIXED(15),
                STARSEN BINARY FIXED(15),
                TEST BIT(1) INITIAL('0'B),
                TWOSNT BINARY FIXED(15),
                TYPE CHARACTER(3) VARYING;
```

```
RANDOM:
      PROCEDURE(LOW,HIGH) RETURNS( BINARY FIXED(15));
           DECLARE LOW BINARY FIXED(15),
                   HIGH   BINARY FIXED(15),
                   PRIME  BINARY FIXED(15)
                          INITIAL(127),
                   MODULO BINARY FIXED(15)
                          INITIAL(256),
                   RESULT BINARY FIXED(15);

           SEQUENC=MOD(SEQUENC*PRIME+1,MODULO);
           RESULT = ABS(MOD(SEQUENC,HIGH-LOW+1))+LOW;
           IF DEBUG THEN DO;
                      CALL PUTCR
                        (TWO,'SEQUENC='||DEC(SEQUENC));
                      CALL PUTCR
                        (TWO,'RESULT='||DEC(RESULT));
                    END;
           RETURN(RESULT);
    END;

    SHIFT: PROCEDURE(BOTTOM,RANGE);
           DECLARE BOTTOM BINARY FIXED(15),
                   K BINARY FIXED(15),
                   RANGE BINARY FIXED(15),
                   TOP BINARY FIXED(15);
           TOP = BOTTOM + RANGE - 1;
           IF DEBUG THEN
              CALL PUTCR (TWO,'BOTTOM='||DEC(BOTTOM)||
                              'TOP='||DEC(TOP));
           DO K = BOTTOM TO TOP;
                  POINT(K) = POINT (K + 1);
                  IF DEBUG THEN
                  CALL PUTCR(TWO,'K='||DEC(K)||
                             ' POINT='||DEC(POINT(K)));
           END;
    END;
```

```
INITF:PROCEDURE(DATA,NAME,TYPE);
  DECLARE DATA CHARACTER(14) VARYING,
          I BINARY FIXED(15),
          NAME CHARACTER(10) VARYING,
          TYPE CHARACTER(3) VARYING;
  I = INDEX(DATA,'.');
  IF DEBUG THEN CALL PUTCR(TWO,'DATA='||DATA||
                                ' I= '||DEC(I));
     IF I = 0  THEN DO;
                 TYPE = '   ';
                 IF LENGTH(DATA) <= 8
                    THEN NAME = DATA;
                    ELSE NAME = SUBSTR(DATA,1,8);
               END;
               ELSE DO;
                 TYPE = SUBSTR(DATA||'   ',I+1,3);
                 NAME = SUBSTR(DATA,1,I-1);
               END;
END;
```

```
RDIN:PROCEDURE(FILEIN,SENTMX,TWOSNT);
  DECLARE FILEIN BINARY FIXED(15),
          LINEMX BINARY FIXED(15),
          SENLIM BINARY FIXED(15) INITIAL(24),
          SENTMX BINARY FIXED(15),
          TWOSNT BINARY FIXED(15);
  TWOSNT = 2;
  LINEMX = 1;
  INITLN(1)= 1;
  DO SENTMX = 2 BY 2 TO SENLIM;
     INITLN(SENTMX) = 1;
  END;
  SENTMX = 1;
  CALL GET(FILEIN,HEAD(1),RC);
  DO WHILE ( RC ^= 2);
     IF RC ^= 2 THEN DO;
        CALL GET(FILEIN,EXTEXT(LINEMX),RC);
        LINEMX = LINEMX + 1;
        IF LINEMX > MXSIZE THEN DO;
           CALL PUTCR(TWO,
              ' SENTENCES EXCEED ALLOWED STORAGE');
           RC = 2;
        END;
     END;
     IF RC ^= 2
        THEN CALL GET (FILEIN, HEAD(SENTMX+1), RC);
     IF RC ^= 2
        THEN DO;
          IF HEAD(SENTMX+1) = '  '
             THEN INITLN(TWOSNT) = INITLN(TWOSNT) + 1;
             ELSE DO;
               TWOSNT = TWOSNT + 2;
               INITLN(TWOSNT - 1) = LINEMX;
               SENTMX = SENTMX + 1;
             END;
        END;
  END;
END;
```

```
QRDIN:PROCEDURE(FILE2);
     DECLARE I BINARY FIXED(15),
             FILE2 BINARY FIXED(15),
             LINEMX BINARY FIXED(15),
             NLPQ BINARY FIXED(15),
             NS BINARY FIXED(15),
             NQ BINARY FIXED(15),
             TNQ BINARY FIXED(15),
             TQUEST CHARACTER(2) VARYING;
   LINEMX = 1;
   NLPQ = 1;
   NS = 2;
   NQ = 0;
   TNQ = 1;
   QSIZE(1) = 1;
   CALL GET (FILE2,QUESTID(1),RC);
   CALL GET (FILE2,TQUEST,RC);
   DO WHILE (RC~=2);
      CALL GET (FILE2,RESTLN(LINEMX),RC);
      LINEMX = LINEMX + 1;
      IF LINEMX > QMAX THEN DO;
         CALL PUTCR (TWO,
              ' QUESTIONS EXCEED STORAGE CAPACITY ');
         RC = 2;
      END;
      IF RC ~= 2
         THEN CALL GET (FILE2,QUESTID(NS + 1),RC);
      IF RC ~= 2
         THEN CALL GET (FILE2,TQUEST,RC);
      IF (TQUEST ~= ' ') | (RC = 2) THEN DO;
         TNQ = TNQ + 1;
         QSIZE(TNQ) = NLPQ;
         IF TEST
            THEN CALL PUTCR(TWO, DEC(TNQ)||
                                 ' NLPQ = '||DEC(NLPQ));
         NQ = NQ + 1;
         NLPQ = 1;
       END;
      ELSE NLPQ = NLPQ + 1;
```

```
            IF (QUESTID(NS + 1) ~= ' ') | (RC = 2) THEN DO;
               QUESTID(NS) = DEC(NQ);
               TNQ = NS * 3 + 1;
               IF RC ~= 2 THEN QSIZE(TNQ) = LINEMX;
               IF TEST THEN DO;
                 DO I = 1 TO NS;
                  CALL PUT (TWO,QUESTID(I));
                 END;
                 CALL PUTCR (TWO,
                             ' THIS IS THE QUESTID ARRAY ');
                 DO I = 1 TO TNQ-1;
                    CALL PUT (TWO, DEC(QSIZE(I)));
                 END;
                 CALL PUTCR (TWO,
                             ' THIS IS THE QSIZE ARRAY ');
                 CALL PUTCR (TWO,' NS NQ LN '||
                             DEC(NS)||DEC(NQ)||DEC(LINEMX));
               END;
               NS = NS + 2;
               NQ = 0;
             END;
           END;
     END;

CALL PUT (TWO,' TYPE THE NAME OF THE FILE OF SENTENCES,'||
             ' THEN PRESS RETURN ');
CALL GET (IN,SENTNM,RC);
CALL PUTCR (TWO,' ');
CALL PUT (TWO,
         ' ENTER LETTER FOR FIRST TYPE OF DATA (E OR P)');
CALL GET (IN,ANSWER,RC);
LETTER(1) = ANSWER;
IF ANSWER = 'E' THEN LETTER(2) = 'P';
               ELSE LETTER(2) = 'E';
CALL PUTCR(TWO,' ');
IF DEBUG THEN CALL PUTCR(TWO,DATA);
CALL PUT (TWO,
         ' GIVE AN INITIAL THREE DIGIT RANDOM NUMBER ');
CALL GET(IN,TYPE,RC);
SEQUENC = BIN(TYPE);
CALL PUT (TWO,
         ' TYPE THE NUMBER OF THE FIRST SENTENCE'||
         ' THEN PRESS RETURN ');
CALL GET (IN,TYPE,RC);
STARSEN = BIN(TYPE);
IF DEBUG THEN CALL PUTCR(TWO,NAME||TYPE);
CALL PUT (TWO,
         ' TYPE IN YOUR LAST NAME THEN PRESS RETURN ');
CALL GET (IN,DATA,RC);
CALL PUT (TWO,' TYPE IN YOUR SESSION NUMBER (1 OR 2) ');
```

```
CALL GET (IN,ANSWER,RC);
CALL INITF(DATA,NAME,TYPE);
NAME = ANSWER || NAME;
CALL OPEN (CONSOLE,1,NAME,TYPE);
CALL PUTCR(CONSOLE,NAME);
DO LOOP = 1 TO 2;
   CALL PUTCR
        (CONSOLE,SENTNM || LETTER(LOOP) || DEC(STARSEN));
   DATA = SENTNM;
   CALL INITF(DATA,NAME,TYPE);
   NAME = NAME || LETTER(LOOP);
   CALL OPEN(FILEIN,0,NAME,TYPE);
   CALL RDIN (FILEIN,SENTMX,TWOSNT);
   NAME = NAME || 'Q';
   CALL OPEN(FILE2,0,NAME,TYPE);
   CALL QRDIN(FILE2);
   DO I = 1 TO SENTMX;
         POINT(I) = I;
         IF DEBUG
            THEN CALL PUTCR(TWO,
                    ' POINTER TO HEADING IS '||HEAD(I));
   END;
   CALL PUTCR(TWO,' ');
   NEWMX = SENTMX;
   SETNO = STARSEN;
   DO WHILE (NEWMX > 0);
         CALL PUTCR(TWO,' ');
         CALL PUTCR(TWO,' ');
         IF DEBUG THEN
            CALL PUTCR(TWO,
                       'NEWMX='||DEC(NEWMX)||
                       'SETNO='||DEC(SETNO));
         SIZE = (POINT(SETNO) - 1) * 2 + 1;
         IF DEBUG THEN CALL PUTCR(TWO,'SIZE='||DEC(SIZE));
         FIRST = INITLN(SIZE);
         LAST = FIRST + INITLN(SIZE + 1) - 1;
         IF DEBUG
            THEN CALL PUTCR(TWO,
                            'FIRST='||DEC(FIRST)||
                            'LAST'||DEC(LAST));
         DO I = FIRST TO LAST;
            CALL PUTCR(TWO,EXTEXT(I));
         END;
         DO I = LAST + 1 TO FIRST + 22;
            CALL PUTCR(TWO,' ');
         END;
         I = SENTMX - NEWMX + 1;
         CALL PUTCR
             (CONSOLE,' SENTENCE NO '||DEC(I)||
                      ' ID NO. '||QUESTID(SIZE));
```

```
NSENT = BIN (QUESTID(SIZE+1));
CALL STM;
CALL PUT(TWO,' PRESS RETURN WHEN YOU THINK'||
             ' YOU CAN REMEMBER THIS PROGRAM ');
ANSWER = CHARIN;
CALL GTM;
CALL PUTCR(TWO,ANSWER);
ORDER(I) = HEAD(POINT(SETNO));
CALL PUTCR
  (CONSOLE,' ID NO.'||ORDER(I)||
           ' NO OF LINES'||DEC(INITLN(SIZE+1)));
INDX = (POINT(SETNO) - 1 ) * 6 + 1;
IF TEST THEN
   CALL PUTCR(TWO,QUESTID(SIZE)||
                  QUESTID(SIZE + 1)||DEC(INDX));
FIRST = QSIZE(INDX);
DO K = 1 TO NSENT;
   LAST = FIRST + QSIZE(INDX + K) - 1;
   IF TEST THEN
      CALL PUTCR(TWO,' FIRST '||DEC(FIRST)||
                     ' LAST '||DEC(LAST));
   DO I = FIRST TO LAST;
      CALL PUTCR (TWO,' ');
      CALL PUTCR (TWO,RESTLN(I));
   END;
   DO I = FIRST + (LAST - FIRST + 1) * 2
        TO FIRST + 22;
      CALL PUTCR (TWO,' ');
   END;
   CALL PUT (TWO,
      ' PRESS THE LETTER WHICH CORRESPONDS'||
      ' TO THE CORRECT ANSWER ');
   CALL STM;
   ANSWER = CHARIN;
   CALL GTM;
   CALL PUTCR (TWO,' ');
   CALL PUTCR (CONSOLE,
        ' ANSWER TO QUESTION NO. '||DEC(K)||
        ' IS '||ANSWER);
   FIRST = LAST + 1;
END;
```

```
            IF NEWMX > 0 THEN DO;
               CALL SHIFT(SETNO,NEWMX - SETNO);
               IF DEBUG THEN DO;
                  DO I = 1 TO SENTMX;
                     CALL PUT(TWO,DEC(POINT(I)));
                  END;
               END;
            NEWMX = NEWMX - 1;
            IF NEWMX > 0 THEN SETNO = RANDOM(1,NEWMX);
         END;
      END;
  STARSEN = MOD(STARSEN + 2, 6) + 1;
  END;
  CALL CLOSE(CONSOLE);
  CALL PUTCR(TWO,' ');
  CALL PUTCR(TWO,
       ' THANK YOU FOR PARTICIPATING IN THIS EXPERIMENT.');
  CALL PUTCR(TWO,
       ' THIS CONCLUDES THIS PORTION OF THE TASK.);
END;

STM: PROCEDURE;
%INCLUDE GLOBAL1;

   CALL WRTM(2,1440);
   CALL WRTM(1,6000);
   CALL WRTM(0,20000);
END; /* OF MAIN */

GTM: PROCEDURE;
%INCLUDE GLOBAL1;
  DECLARE DEBUG BIT(1) INITIAL('0'B),
          WHICH BINARY FIXED(15),
          WVL   BINARY FIXED(15),
          RVL0 BINARY FIXED(15),
          RVL1 BINARY FIXED(15),
          RVL2 BINARY FIXED(15),
          MIN BINARY FIXED(15),
          SEC BINARY FIXED(15),
          MS  BINARY FIXED(15),
          RTM BINARY FIXED(15),
          MMS BINARY FIXED(15);

  RTM=0;
  IF DEBUG THEN CALL PUTCR(TWO,'RTM='||DEC(RTM));
  RVL2=RDTM(2);
  IF DEBUG THEN CALL PUTCR(TWO,'RVL2='||DEC(RVL2));
  RVL1=RDTM(1);
  IF DEBUG THEN CALL PUTCR(TWO,'RVL1='||DEC(RVL1));
```

```
    IF RVL1=6000
    THEN DO;
      RVL2=RDTM(2);
      RTM=RTM+1;
    END;
    RVL0=RDTM(0);
    IF DEBUG THEN CALL PUTCR(TWO,'RVL0='||DEC(RVL0));
    IF RVL0 > 19950
    THEN DO;
      RVL1=RDTM(1);
      RTM=RTM+1;
    END;
    RVL0=RVL0+(RTM+2)*44+364;
    IF RVL0 > 20000
    THEN DO;
    RVL0=RVL0-20000;
    RVL1=RVL1+1;
    END;
    MIN=1440-RVL2;
    SEC=(6000-RVL1)/100;
    MS= (20000-RVL0)/2000;
    MMS=(20000-RVL0-MS*2000)/2;
    MS=MS+(6000-RVL1-100*SEC)*10;
    CALL PUT(CONSOLE,'TIME: ');
    CALL PUT(CONSOLE,DEC(MIN));
    CALL PUT(CONSOLE,' MIN');
    CALL PUT(CONSOLE,DEC(SEC));
    CALL PUT(CONSOLE,' SEC');
    CALL PUT(CONSOLE,DEC(MS));
    CALL PUT(CONSOLE,' MS');
    CALL PUT(CONSOLE,DEC(MMS));
    CALL PUT(CONSOLE,' MMS');
END; /* OF MAIN */
```

BIBLIOGRAPHY

BIBLIOGRAPHY

Adelson, B. (1981). Problem solving and the development of abstract categories in programming. Memory and Cognition, 9, 422-433.

Adelson, B. (1983). Structure and strategy in the semantically rich domains. Dissertation Abstracts International, (University Microfilms No. 8322293).

Anderson, J. R., and Bower, G. H. (1973). Associative Memory. V. H. Winston and Sons, Washington, D.C.

Anderson, J. R. (1974A). Verbatim and propositional representation of sentences in immediate and long-term memory. Journal of Verbal Learning and Verbal Behavior, 13, 149-62.

Anderson, J. R. (1974B). Retrieval of propositional information from long-term memory. Cognitive Psychology, 6, 451-71.

Anderson, J. R. and Reder, L. M. (1979). An elaborative processing explanation of depth of processing. In Cermak, L. S. and Craik, F. I. M. (Eds.) Levels of Processing in Human Memory, Hillsdale, N. J.: Lawrence Erlbaum Associates.

Anderson, R. C., Spiro, R. J., and Anderson, M. C. (1978). Schemata as scafolding for the representation of information in connected discourse. American Eduactional Research Journal, 15, 443-440.

Anderson, R. C. (1984). Some reflections on the acquisition of knowledge. Educational Researcher, 13, 5-10.

Atwood, Michael E. and Ramsey, Rudy, H. (1978). Cognitive structures in the comprehension and memory of computer programs: An investigation of computer debugging, (Technical Report, T.R.-78-A210), Science Applications, Englewood Colo.

Atwood, Michael E., Ramsey, H. R., Hooper, J. N. and Kullas, D.A. (1979). Annoted bibliography on human factors and software development. (ARI Technical Report), U. S. Army Research Institute, Alexandria Virginia.

Barnard, P. J. Hammond, N. V., Morton, J., Long, S. B., and Clark I. A. (1981). Consistency and compatibility in human and computer dialogue. International Journal of Man-Machine studies, 15, 87-134.

Biermann, A. W., Ballard, B. W. and Holler, A. (1979). An Experimental study of natural language programming. (Report CS-1979-9), Duke University, Durham North Carolina.

Bierman, A. W., and Ballard, B. W. (1980). Toward natural language computation. American Journal of Computational Linguistics, 6, 71-86.

Black, J. B. and Bern, H. (1981). Casual coherence and memory for events in narratives. Journal of Verbal Learning and Verbal Behavior, 20, 267-275.

Black, J. B. and Sebrechts, M. M., (1981). Facilitating human-computer communication. Applied Psycholinguistics, 2, 149-177.

Black, M. C. (1983). Readability as an interactive process: an analysis of the syntactic and semantic effects on text of cloze errors associated with variation in the structure of text and in the readers' knowledge of text topic. Dissertation Abstracts International, (University Microfilms No. 8321292).

Black, J. B. and Moran, T. P. (1982). Learning and remembering command names. Proceedings of Human Factors in Computing Systems, 8-11.

Bransford, J. D. and Franks, J. J. (1972). The abstraction of linguistic ideas. Cognitive Psychology, 2, 331-350.

Bransford, J. D. and Johnson, M. K. (1972). Contextual prerequisites for understanding: some investigations of comprehension and recall. Journal of Verbal Learning and Verbal Behavior, 11, 717-726.

Bransford, J. D. and Johnson, M. K. (1973). Considerations of some problems of comprehension. In W. G. Chase (Ed.), Visual Information Processing. New York: Academic Press.

Britton, B. K., Westbook, R. D. and Holdredge, T. S. (1978). Reading and cognitive capacity usage: effects of text difficulty. Journal of Experimental Psychology: Human Learning and Memory, 4 582-591.

Brooks, R. (1975). A model of human cognitive behavior in writing code for computer programs. Proceedings of the Fourth International Joint conference of Artificial Intelligence.

Brooks, R. (1977). Towards a theory of the cognitive processes in computer programming. International Journal of Man-Machine Studies, 9, 737-751.

Carroll, J. M. (1980). Learning, using, and designing command paradigms. (IBM Research Report RC 8141).

Chase, W. G. and Simon H. A. (1974). Perception in chess, Cognitive Psychology, 4, 55-81.

Chrysler, E. (1978). Some basic determinants of computer programming productivity. Communications of the ACM, 21, 581-587.

Chrysler, E. (1978). The impact of program and programmer characterisics, Proceedings of the National Computer Conference, AFIPS Press, Montvale, New Jersey, 47, 581-587.

Clark, H. H. and Card, S. K. (1969). Role of semantics in remembering comparative sentences. Journal of Experimental Psychology, 82, 545-553.

Clark, H. H. and Clark, E. V. (1977). Psychology and Language. New York: Harcourt, Brace Jovanaovich.

Cooke, J. E. and Bunt, R. R. (1975). The need to study the individual programmer. Infor, 13, 296-307.

Cromer, W. (1970). The difference model: A new explanation for some reading difficulties. Journal of Educational Psychology, 61, 471-483.

Curtis, B. (1979). In search of software complexity. Proceedings of the Workshop on Quantitative Models of Software Reliability, Complexity and Cost. New York: Institute of Electrical and Electronics Engineers.

Curtis, B., Sheppard S. B., Milliman P., Borst M. A. and Love, T. (1979). Measuring the psychological complexity of software maintenance tasks with Halstead and McCabe metrics. IEEE Transactions on Software Engineering (SE-5) 2, 96-104.

Dijkstra, E. W. (1963). On the design of machine independent programming languages. in Annual Review in Automatic Programming (Vol. 3). R. Goodman (ed.) New York: pergamon Press, 27-33.

Dijkstra, E. W. (1964). Some comments on the aims of MIRFAC. Communications of the ACM, 7, 190.

Dixon, P. (1980). Following written instructions. Manuscript, Bell Laboratories. Murray Hill, New Jersey.

Dooling, D.J. and Lachman, R. (1971). Effects of comprehension on retention of prose. Journal of Experimental Psychology, 88, 216-222.

Dyck, J. L. and Mayer, R. C. (1985). BASIC versus natural language: Is there one underlying comprehension process? Proceedings CHI'85 Human Factors in Computing Systems, San Francisco, April 14-18, ACM, New York, 221-224.

Embly, D. W. (1975). An experiment on a unified control construct (Technical Report No. UIUCDCS-R-75-759). Department of Computer Science, University of Illinois, Urbana Illinois.

Embly, D. W. (1976). Experimental and formal language design applied to control constructs for interactive computing (Technical Report No. UIUCDS-R-76-811). Department of Computer Science, University of Illinois, Urbanna, Illinois.

Frase, L. T. (1978). Inference and reading. In Revlin, R. and Mayer, R. E. (Eds.), Human Reasoning. New York: Wiley.

Frederikson, C. H. (1972). Effects of task-induced cognitive operations on comprehension and memory processes. In R. D. Freedle and S. B. carroll (Eds.) Language comprehension and the acquisition of knowledge. New York: Winston.

Frost, D. (1975). Psychology and program design. Datamation, 5, 137-138.

Gagnè, E. D., Yarbrough, D. B., Bell, M., and Weidemann, C. (1984). Does familiarity have an effect on recall independent of its effect on original learning? Unpublished Paper, University of Georgia.

Gannon, J. D. and Horning, J. J. (1975). The impact of language design on the production of reliable software. IEEE Transactions on Software Engineering (SE-1), 2, 179-191.

Gannon, J. D. (1976). An experiment for the evaluation of language features. International Journal of Man-Machine Studies, 8, 61-73.

Gannon, J. D. (1977). An experimental evaluation of data type conventions. Communications of the ACM, 20, 584-595.

Garrod, S. and Trabasso, T. A. (1973). A dual-memory information processing interpretation of sentence comprehension. Journal of Verbal learning and Verbal Behavior,12, 155-167.

Gillespie, E. R. (1983). Effects of levels of propositional and syntactic complexity upon reading comprehension. Dissertation Abstracts International, (University Microfilms No. 8326172).

Gould, J. D. and Drongowski, T. (1974). An Exploratory study of computer programming debugging. Human Factors, 16, 258-277.

Graesser, A. C. Hauft-Smith, K., Cohen, A. D., and Pyles, L. D. (1980). Advanced outlines, familiarity text genre, and retention of prose. Journal of Experimental Education, 48, 209-220.

Graesser, A. C., Hoffman, N.L., and Clark, L.F. (1980). Structural components of reading time. Journal of Verbal Learning and Verbal Behavior, 19, 131-151.

Graesser, A. C. (1981). Prose Comprehension beyond the word. New York, N.Y.: Springer-Verlag.

Green, T. R. G., (1977). Conditional Program Statements and their comprehensibility to professional programmers. Journal of Occupational Psychology, 50, 93-109.

Greenblatt, D. and Waxman, J. (1978). A study of three database query languages In B. Shneiderman (Ed.), Database Improving Usability and Responsiveness. Academic Press, 77-97.

Guiliano, V. E. (1972). In defense of natural language. Proceedings of the Association of Computing Machinery National Conference. (1074) New York: ACM.

Halpern, M. (1967). Foundations of the case for natural-language programming. IEEE Spectrum, 4, 140-149.

Herriot, R. (1977). Towards the ideal programming language. SIGPLAN Notices, 11, 56-62.

Hill, I. D. (1972). Wouldn't it be nice if we could write computer programs in ordinary English - or would it? Computer Bulletin, 16, 306 - 312.

Hobbs, J. R. (1977). What the nature of natural language tells us about how to make natural-language-like programming languages more natural. SIGPLAN Notices, 12 85 - 93.

Hoc, J. M. (1977). The role of mental representation in learning a programming language. International Journal of Man-Machine Studies, 9, 87-105.

Hsu, R. W. (1978). Grammatical function and readability. Proceedings Hawaii International Conference in System Sciences, 5-14.

Johnson, R. G. (1970). Recall of prose as a function of the structural importance of the linguistic units. Journal of Verbal Learning and Verbal Behavior, 9, 12-20.

Kammann R. (1972). The comprehensibility of printed instructions and the flow chart alternative. Human Factors, 17, 183-191.

Kernighan and Plaugher (1978). The elements of programming style (2nd. ed.) New York: McGraw Hill.

Kintsch, W. and Monk, D. (1972). Storage of complex information in memory: some implications of the speed with which inferences can be made. Journal of Experimental Psychology, 94, 25-32.

Kintsch, W. and Keenan, J. M. (1973). Reading rate and retention as a function of the number of propositions in the base structure of sentences. Cognitive Psychology, 5, 257-274.

Kintsch, W. (1974). The representation of meaning in memory. Hillsdale, N.J.: Erlbaum.

Kintsch, W., Kozminsky, E., Streby, W. J., Mckoon, G., and Keenan, J. M. (1975). Comprehension and recall of text as a function of content variables. Journal of Verbal Learning and Verbal Behavior, 14, 196-214.

Kintsch, W. (1977). Memory and Cognition. New York: Wiley.

Kintsch, W. and van Dijk, T. A. (1978). Toward a model of text comprehension and production. Psychological Review, 85, 363-394.

Kintsch, W. (1979). On modeling comprehension. Educational Psychologist, 14, 3-14.

Kintsch, W. and Vipond, D. (1979). Reading comprehension and readability in educational practice and psychological theory. In Nilsson, L. G. (Ed.) Perspectives on Memory Research. Hillsdale, N. J.: Erlbaum.

Love, T. (1977). Relating individual differences in computer programming performance to human information processing abilities. Dissertation Abstracts International, (Xerox University Microfilms No. 77-18,379).

Manelis, L. and Yekovich, F. R. (1976). Repetitions of propositional arguments in sentences. Journal of Verbal Learning and Verbal Behavior, 15, 301-312.

Marks, L.E., and Miller, G. A. (1964). The role of semantic and syntactic constructs on the memorization of English sentences. Journal of Verbal Learning and Verbal Behavior, 3, 1-5.

Mayer, R. E. (1979). A psychology of learning BASIC. Communications of the ACM, 22, 589-593.

McKeithen, K. B. and Reitman, J. S. (1981). Knowledge organization and skill differences in computer programs. Cognitive Psychology, 13, 307-325.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review, 63, 81-97.

Miller, L. A. (1974). Programming by non-programmers. International Journal of Man-Machine Studies, 6, 237-260.

Miller, L. A. (1975). Naive programmer problems with specification of transfer-of-control. Proceedings of the National Computer Conference, AFIPS Press, Montvale, New Jersey, 44, 657-663.

Miller, L. A. (1978). Behavioral studies of the programming process (IBM Research Report, RC 7367).

Miller, L. A.(1981). Natural language programming: styles, strategies, and contrasts. IBM Systems Journal, 20, 184-215.

Moeser, S. D. (1975). Memory for language organization in concrete and abstract sentences. Memory and Cognition, 5, 560-568.

Montgomery, C. A. (1972). Is Natural Language an unnatural query language? Proceedings of the ACM Conference, ACM, New York, 1075-1078.

Myers, G. J. (1978). A controlled experiment in program testing and code walkthroughs/inspections. Communications of the ACM, 21, 760 - 768.

Norman, D. A. and Rumelhart, D.S. (1975). Explorations in Cognition. San Francisco.

Norman, D. A. (1982). Learning and Memory. San Francisco: W. H. Freeman and Company.

Peterson, W. W. (1981). Private Communication.

Petrick, S. R. (1976). On natural language based computer systems. IBM Journal of Research and Development, 20, 314 - 325.

Reeker, L. H. (1980). Natural language programming and natural programming languages. The Australian Computer Journal, 12, 89-92.

Reisner, P. (1977). Use of Psychological experimentation as an aid to development of a query language. IEEE Transactions on Software Engineering (SE-3), 218-229.

Riley, P. M. (1973). The cloze procedure -- a selected bibliography. (ERIC Document Reproduction Service No. ED 106 749)

Roberts, T. L. (1979). Evaluation of computer text editors. (Report SSL-79-9), Xerox Palo Alto Research Center, Palo Alto, California.

Royer, J. M., Lynch, D. J., Hambleton, R. K., and Bulgareli, C. (1984). Using the sentence verification technique to asses the comprehension of technical text as a function of subject matter expertise. American Educational Research Journal, 21, 839-869.

Rumelhart, D. E. (1977). Toward an interactive model of reading. In Dornic, S. (Ed.), Attention and performance VI. Hillsdale, N. J.: Erlbaum.

Sammet, J. M. (1966). The use of English as a programming language. Communications of the ACM, 9, 228-230.

Sheil, B. A. (1981). The psychological study of programming. ACM Computing Surveys, 13, 101-120.

Sheppard, S. B., Kruisi, E., Bailey, J. W. (1982). An empirical evaluation of software documentation formats. Proceedings Human Factors in Computer Systems, 121-124.

Shneiderman, B. (1976). Exploratory experiments in programmer behavior. Int. J. Comput. Inf. Sci., 5, 123-143.

Shneiderman, B. (1977). Measuring computer program quality and comprehension. International Journal of Man-Machine Studies, 9, 465-478.

Shneiderman, B. and Mayer, R. (1979). Syntactic/semantic interactions in programmer behavior: A model and experimental results, Int. J. Comput. Inf. Sci., 7, 219-239.

Shneiderman, B. (1980). Software Psychology: Human Factors in Computer and Information Systems. Cambridge, Mass.: Winthrop Publishers, 1980.

Sime, M. E., Green, T. R. G., and Guest, D. J. (1973). Psychological evaluation of two conditional constructions used in computer languages. International Journal of Man-Machine Studies, 5, 105-113.

Sime, M. E., Green, T. R. G., and Guest, D. J. (1977). Scope matching in computer conditionals-a psychological evaluation. International Journal of Man-Machine Studies, 9, 107-118.

Simmons, R. F. (1986). Man-machine interfaces: can they guess what you want? IEEE Expert, 1, 86-94.

Small, D. W. and Weldon, L. J. (1983). An experimental comparison of natural and structure querry languages. Human Factors, 25, 253-263.

Soloway, E., Bonar, J., and Ehrlich, K. (1983). Cognitive Strategies and Looping Constructs: An Empirical Study. Communications of the ACM, 1983, 26, 853-860.

Spilich, G. J., Vesonder, G. T., and Chiesi, H. L., and Voss, J. F. (1979). Text processing of domain related information for individuals with high and low domain knowledge. Journal of Verbal Learning and Verbal Behavior, 18, 275-290.

Stevens, A. and Rumelhart, D. E. (1975). Errors in reading: Analysis using an augmented transition network model of grammar. In D.A. Norman and Rumelhart, D. E. (Eds.). Explorations in Cognition. San Francisco, Freeman.

Vipond, D. (1980). Micro- and macroprocesses in text comprehension: Studies in reminding and resembling. Journal of Verbal Learning and Verbal Behavior, 19, 276-296.

Weissman, L. M. (1974). A Methodology for studying the psychological complexity of computer programs. (Technical Report 37). Toronto, Ontario, Canada: University of Toronto, Computer Systems Group.