PREDICTING PROGRAM COMPLEXITY FROM WARNIER-ORR DIAGRAMS

by

BARBARA WHITE

B. A., University of Kansas, 1965
M. A., University of Missouri, 1968

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:

Virg Wallentine

Major Professor

TABLE OF CONTENTS

Page

CHAPTER 1 APPLICATION OF HALSTEAD'S COMPLEXITY MEASURES TO PROGRAM DESIGN 1
1.1 Introduction to the Purpose of the Experiment 1
1.2 Derivation of Halstead's Formulas 2
1.3 Published Studies of the Practical Applications of Halstead's Theories 10
1.3.1 Fitzsimmons and Love's Review of Software Science 10
1.3.2 Christensen et al.'s Study of Halstead's Metrics and Program Design 11
1.4 Warnier-Orr Diagrams 14
1.4.1 Warnier-Orr Design Methodology 16
1.4.2 Theory of Warnier-Orr Structures 19
1.4.3 Present Usage of the Warnier-Orr Technique 20
1.4.4 STRUCTURE(S): An Automated Warnier-Orr Diagram Drawing Package 20

CHAPTER 2 AN OPERANDS AND OPERATOR COUNTING TECHNIQUE FOR WARNIER-ORR DIAGRAMS 23
2.1 Experimental Assumptions 23
2.2 Restriction on STRUCTURE(S)-Style Input Lists for Program COUNT 24
2.3 Structure of Program COUNT 26
2.4 Program Operator and Operand Counting Program 29

CHAPTER 3 PREDICTIVE POWER OF HALSTEAD DESIGN VALUES FOR PROGRAM VALUES 31
3.1 Experimental Hypothesis 31
3.2 Experimental Procedure 32
3.3 Results 33
3.3.1 Validity of the Diagram Operator and Operand Counting Technique 33
3.3.2 Diagram:Program Ratios of the Halstead Metrics 35
3.3.3 The V* Metric 43
3.4 Conclusions 43

REFERENCES 46

Page

APPENDIX A A-1
APPENDIX B B-1
APPENDIX C C-1
APPENDIX D D-1
APPENDIX E E-1
APPENDIX F F-1
APPENDIX G G-1
APPENDIX H H-1

LIST OF TABLES

Page

Table 3.1 Relationship of Estimated Length to Actual Length for Diagrams and Programs 36

Table 3.2 Volume and Volume Ratio for Diagrams and Programs 37

Table 3.3 Language Level and Language Level Ratio for Diagrams and Programs 38

Table 3.4 Estimated Abstraction Level, Difficulty, Structure, and Abstraction Level Ratios for Diagrams and Programs 39

Table 3.5 Mental Effort, Time, and Mental Effort Ratios for Diagrams and Programs 40

LIST OF FIGURES

		Page
Figure 1	Higgins, 1979a, p. 2	15
Figure 2	Higgins, 1979b, p. 191	17
Figure 3	Higgins, 1879b, pp. 120-121	19
Figure 4	Syntex diagrams for Warnier-Orr basic structures as implemented in this study using STRUCTURE(S) source input lists.	28

CHAPTER 1

APPLICATION OF HALSTEAD'S COMPLEXITY MEASURES TO PROGRAM DESIGN

1.1 Introduction to the Purpose of the Experiment

Warnier-Orr diagrams are the product of a program design technique invented by Jean-Dominique Warnier and extended by Kenneth Orr that is claimed to be far superior to other techniques such as flowcharting. A WO diagram is always structured and is progressively expandable as the program designer refines his work to the point where it is coded in a programming language. Kenneth Orr has marketed a commercial version of the Warnier-Orr technique, called STRUCTURE(S) (Langston Kitch, 1978), that produces printouts of WO diagrams; it is meant for use as a tool for the design of large, complex systems programs.

Maurice Halstead, on the other hand, was the inventor of a language-independent software metrics that he claimed to reveal the inherent complexity of a program; he is the father of software science, and a large number of studies have attempted to validate his theories. In spite of the difficulty of discovering the mathematical basis for Halstead's equations, they have been shown to be accurate predictors of such factors as number of program errors and time required to produce programs (Fitzsimmons and Love, 1978, p. 5).

One reason for continuing interest in Halstead's program complexity metrics is the possibility of applying them to practical problems in designing and coding. For example, Christensen, Fitsos, and Smith, in a review and analysis of software science, say that Halstead's complexity measures--i.e., on the first clean compile of a program--and that it is highly desirable "to use measurements that can lead to the optimization of program organization while the program is being written or while it is being designed. . . . Software engineering. . . needs a measurement discipline that each programmer can understand and can relate to choices made while designing and coding a program" (Christensen et al., 1981, p. 373).

If, in fact, Halstead's metrics were to prove applicable to a stage of program design considerably earlier than the first clean Warnier-Orr diagrams--and to predict reasonably well the complexity of program written from the diagrams, then one more very useful feature would have been added to the WO diagram technique. As well, such results would tend to substantiate further the language-independent nature of Halstead's software science theories beyond the programming language realm.

These propositions were the motivation for the present study of ways to adapt Halstead's measurement techniques to a structured design language and of the resulting relationships, if any, between Halstead values for designs and those for programs based on the designs.

1.2 <u>Derivation of Halstead's Formulas</u>

Because Halstead's theories represent a novel approach to the definition and analysis of program complexity, they require a fairly elaborate explanation. Fortunately, Halstead himself took pains to make his derivations widely available. In Volume 18 of <u>Advances in Computers</u>, Halstead (1979a) defines the five components of a science--sound metrics, reproducible experiments, derivable relationships, ability to explain observed phenomena, and ability to predict the result of an experiment. Software he defines as "any communication that appears in symbolic form in conformance with the grammatical rules of any language" (pp. 119-120). The function of software science is to provide the theoretical foundation for software engineering.

Although later in the same article, Halstead discusses at length the various applications of his metrics to technical English prose and to the psychology of reading and writing, his inferences seem to have no particular relevance to linguistic theory, and apparently most published studies of his theories deal with software as defined more conventionally, that is, computer programs.

All of Halstead's equations for measuring complexity are based on counts of operators and operands, the two mutually exclusive entity categories that constitute any computer program in any language. Halstead defines an operand as a variable or a constant and an operator as "an entity that can alter either the value of an operand of the order in which it is altered" (p. 121). His basic measures, from which the others are derived, are N_1, the total occurrences of operators in a program, N_2, the total occurrences of operands, η_1, the number of unique operators, and η_2, the number of unique operands.

The vocabulary of a program is simply

$$\eta = \eta_1 + \eta_2 \qquad (1)$$

and the length is

$$N = N_1 + N_2 \qquad (2)$$

According to Halstead, the concept of program volume is best derived from and N on the basis of the minimum number of bits required to represent each operator and operand multiplied by total occurrences:

$$V = N \log_2 \eta \tag{3}$$

Volume is, in fact, dependent on the language in which a program is written, because a higher-level language can perform a given function in fewer instructions than a lower-level language.

This concept leads to the idea of the highest-level language, for which every result would be available by calling a built-in procedure or function and for which the volume would be smallest. For any program written in the highest-level language, only two operators would be needed, one for the name of the procedure and one to group the operands, of which a variable number would be required depending on the nature of the subroutine. Potential volume is written

$$V^* = N^* \log_2 \eta^*$$

Because no operators or operands would have to be repeated in the highest-level language, $N^* = \eta^*$, so that

$$V^* = \eta^* \log_2 \eta^*$$

In terms of operators and operands, this is

$$V^* = (\eta_1^* + \eta_2^*) \log_2(\eta_1^* + \eta_2^*)$$

and because $\eta_1^* = 2$, potential volume becomes finally

$$V^* = (2 + \eta_2^*) \log_2(2 + \eta_2^*)$$

Representing as it does the minimum volume for an algorithm, the potential volume is language independent.

Halstead derives an equation for "implementation level" defined as the ratio of potential volume to the actual volume of a given implementation:

$$L = V^*/V$$

which means that another way of expressing potential volume is

$$V^* = LV \tag{4}$$

This is the formula for potential volume used in the present study, with est. L substituted for L because, according to Halstead, a close approximation of the actual level may be obtained by assuming that the more unique operators used in an implementation the lower the program level, with the minimum possible being, of course, $_1^* = 2$. Therefore,

$$L \sim \eta_2/N_2$$

Halstead proposes the following equation, because the "simplest combination of these two terms that will meet the condition that L = 1 for a potential language is their product, where the constant of proportionality is one" (p. 124):

$$\text{est. } L = (\eta_1^*/\eta_1)\,(\eta_2/N_2)$$

or

$$\text{est. } L = (2/\eta_1)\,(\eta_2/N_2) \qquad (5)$$

Halstead says that est. L has been proven by experiment to be close enough to L for the former to be used interchangeably with the latter.

Because LV should be a language-independent constant value for a particular program, potential volume is a useful measurement for testing the application of Halstead's theories to program designs and the programs written from them. (However, there is some question whether two versions of a program written in two different languages can ever be exactly the "same" program.)

For different programs written in the same language, the potential volume V^* must increase as program size increases. Halstead says that implementation level L decreases proportionally with the increase in potential volume so that a language level

$$\lambda = LV^*$$

may be defined that "tends to remain nearly constant over a wide range of program sizes" (p. 125). As should be expected if the concept of language level has any meaning, Halstead and others have found that language level increases from lower-level programming languages to higher-level programming languages to technical English prose. Although this increase is consistent, variances are large and grow larger as language level increases, so that there is considerable overlap.

In the present study the equation for language level used is that used by Fitzsimmons and Love (1978, p. 8)

$$\lambda = (\text{est. } L^2)V \qquad (6)$$

based on $V^* = LV$. It was chosen because V can be measured precisely and because Halstead highly recommends the accuracy of est. L.

Halstead's first "counterintuitive" finding in software science was what he calls the vocabulary-length equation:

$$\text{est. } N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \qquad (7)$$

which states that the length of a program may be approximated closely using only its vocabulary. Halstead attempts to explain this formula on the basis that operands and operators tend to alternate in a program and that because a program is "organized" the upper limit of program length must be its logarithm. According to Halstead, a correlation coefficient of greater than 0.98 was obtained for N and est. N in a large series of polished programs.

Because programs can be written whose estimated length is not at all close to the actual N, Halstead determined six "impurity classes" that could account for the discrepancies:

1. Complementary operations--e.g., adding a variable to another and then subtracting it with no intervening logical reason for the operations.

2. Ambiguous operands--e.g., using one variable name to serve different purposes in different parts of a program.

3. Synonymous operands--e.g., using more variable names than are necessary.

4. Common subexpressions--e.g., repeatedly using an expression rather than assigning a name to the result of the expression and using that repeatedly.

5. Unwarranted assignment--e.g., assigning a name to the result of an expression that is used only once.

6. Unfactored espressions--e.g., failing to factor the factorable terms in an expression.

Obviously the impurity classes represent carelessness in programming that should be eliminated by review. However, there are other causes of discrepancy between N and est. N. Christensen et al. (1981, p. 375) reports that one study found est. N to be low for big programs and high for little ones and that another found est. N to be high for 80 percent of a larger number of PL/I programs. In the present study PL/I output formatting statements were found to have a strong confounding effect on est. N if their built-in functions were considered operators on the output variables, and therefore they were eliminated from the counts.

It is also an interesting fact that Halstead's equations show internal consistency when applied to technical English prose, which must be "impure" in order to be readable.

Halstead attempted to determine how hard it must have been for a programmer to write a given program by reasoning that writing a program consists of instituting a binary search through the list of possibilities in the programming language for the N symbols needed. Since each search must require an average of $\log_2$ "elementary mental discriminations," the total is simply the volume of the program:

$$V = N \log_2 \eta$$

which means that mental effort may be defined as volume times number of elementary metal discriminations. And since elementary mental discriminations is a measure of difficulty and abstraction level L can be understood as the inverse of difficulty, a simple representation of mental effort is

$$E = V/L \qquad (8)$$

measured in units of elementary mental discriminations.

As with his other measures, Halstead first found a formula for estimated programming time that worked and then searched out a justification for his empirical result. Equation (9) is based on Halstead's "Stroud rate" of 18 emd's per second, named in honor of John Stroud, a psychologist who estimated that "the human mind is capable of between 5 and 20 mental discriminations per second" (Fitzsimmons and Love, 1978, p. 9):

$$T = E/S \qquad (9)$$

Halstead (1979a, p. 129) says that the rate at which the human brain makes emd's "is nearly constant, and does not vary significantly with intelligence." However desirable an intelligence-independent measure of programming time might be, it is difficult to agree with Halstead that a factor of between 5 and 20 is nearly constant and to understand why 18 is the number of choice other than that it works.

Equation (9) is included in the present study, converted to minutes, and its results are not unreasonable. But, as Fitzsimmons and Love state, Halstead's time equation is in no sense a proof that a programmer took or should have been granted a certain amouunt of time to write a program. (11) However, Halstead (1979a, p. 129) claims his equation to be remarkably accurate in its "ability to predict observed programming times ranging from 5 min to 11,700 man months."

Halstead's speculations about the conclusions that may be drawn from his mental effort hypothesis are wide-ranging. For example, the mental effort value was found to decrease for a program after it had been revised to improve clarity. Someone whose job it was to decide whether a program should be revised might consider whether other programmers than the writer would be assigned to maintain it. If so, and if the mean for the language, the program would seem a likely candidate for revision.

Halstead also discusses the use of his software metrics to predict error rates in programs, the resolution or ambiguities in counting operators, the results of some highly theoretical experiments with his metrics, and the internal consistency of software metrics with respect to technical English prose. Only the last of these discussions is relevant to the purposes of the present study. Halstead's description of how Kulm (1975) and Miller et al. (1958) applied the concept of operators and operands to English provides a starting point for the counting technique used herein for the STRUCTURE(S) design language:

> . . . words were divided into two classes, called function words and content words. The function words, are in general, all of those words that are classified grammatically as articles, pronouns, prepositions, conjunctions, or auxiliary verbs. All of the others are counted as content words . . .
> Kulm reasoned that the content words must be equivalent to operands, and that the function words are operators . . .[to which must be added], of course, the punctuation symbols . . . (Halstead, 1979a, p. 155)

The concept of function words and content words undoubtedly models the structure of the English language. However, a simpler classification of grammatical constructions into verb phrases and noun phrases follows the function word - content word pattern while also in most English grammar textbooks. In the present experiment the prose operators were considered to be verb phrases (e.g., auxiliary and main verbs, infinitive phrases), prepositions, connectives, and punctuation symbols, and the prose operands to be noun phrases (i.e., nouns plus adjectival modifiers not including prepositional phrases).

In the conclusion of his article in Advances in Computers, Halstead invites skepticism of his theories and experimentation with them. He claims that the result will be the "inescapable conclusion" that they tap the natural laws that govern language.

1.3 Published Studies of the Practical Applications of Halstead's Theories

Many large studies of Halstead's theories have been done and have supported with statistics the overall ability of his equations to predict program complexity. Two articles are summarized here in some detail because they indicate the aspects of software science that are currently of interest. The first is a review, and the second is a study of the practical applications of Halstead's equations to program design.

1.3.1 Fitzsimmons and Love's Review of Software Science

Fitzsimmons and Love (1978), in "a review and evaluation of software science," published in Volume 10 of Computing Surveys, outline Halstead's theories much as has been done here already. They list the results of studies that have been done on Halstead's metrics using programs and derive Halstead values for a brief interchange-sort algorithm implemented in Fortran and PDP-11 assembly language.

The computations for their example algorithm come out uniformly well: their 13-line Fortran routine has an N of 50 and an est. N of 52; the volume of the assembly language version of the routine (29 lines) is considerably greater than that of the Fortran version, "because the rich vocabulary of operators in high-level language allows more compact expression and produces shorter programs" (p. 7); the abstraction level is 35 percent higher for the Fortran routine than for the assembler one; the two estimates of V^* agree within 4 percent of each other; and the Fortran routine language level is within one standard deviation of the Fortran average.

Fitzsimmons and Love list mean language level, , and standard deviation for the languages Halstead studied. Those of interest here are

Language	Mean n	S.D.
English prose	2.16	0.86
PL/I	1.53	0.96

1.3.2 Christensen et al.'s Study of Halstead's Metrics and Program Design

"A perspective on software science," by Christensen, Fitsos, and Smith (1981), in Volume 20 of the IBM Systems Journal, discusses the practical uses that might be made of Halstead's metrics in designing a program and in improving it as it is being coded.

Their lists of operator and operand examples and of "some of the not-so-obvious" rules for counting operators were used in the present study for programs and were adapted for use with designs:

> Variable name--operand.
> Literal--operand.
> Arithmetic symbol--operator.
> Punctuation--operator.
> End of statement delimiter--operator.
>
> . . .
>
> Parentheses and brackets always come in pairs, and a compiler diagnoses correct pairing. Each pair is counted as a single "grouping" operator.
> GO TO statements are concatenated with the address of the GO TO to form a single operator.
> If and THEN are combined into a single operator since one is unlikely without the other.
> IF THEN and ELSE are also combined as a single operator. (thus, IF THEN ELSE and IF THEN are two separate and distinct operators.)
> Each of the possible combinations of DO UNTIL, DO WHILE, etc. is combined as a single operator, but each combination is separate from the others. (p. 374)

Another rule perhaps not obvious from Halstead's definitions is that, whether explicit or implied, an end-of-line marker is always counted as present.

Christensen et al. (p. 375) list correlation coefficients for est. N and N from a series of experiments; the relevant ones are the following:

Language	Correlation Coefficient
PL/I	0.98
370 assembler	0.90+
PL/S	0.90+

Programs for the present study were written in PL/I, in UC assembler (which is similar to but smaller than 370 assembler), and in PLDS (like PL/S, a subset of PL/I).

From Halstead's equations and the results of experimentation, Christensen et al. proposes two complexity rules:

1. Program size measured as lines of codes, N, or V is a function of η.

2. For structured programs program size is a function of η_2.

The second rule is based on studies of programs written in PL/S and 370 assembler only and may not be true for all languages; however, it should apply to the programs of the present study.

The difficulty of a program--which as mentioned earlier is the inverse of the implementation or abstraction level (equation 5)--is written

$$D = (\eta_1/2)\ (N_2/\eta_2) \tag{10}$$

Christensen et al. analyzes the separate implications of the two terms on the right-hand side. $\eta_1/2$ refers to the difficulty imposed by a large operator vocabulary, and N_2/η_2 represents the average number of times operands are changed in a program. A higher-level language requires fewer operators, which makes a program easier to write and understand. Frequently changed operands are hard for the reader of a program to keep track of. However, a high difficulty value does not necessarily imply that there is something wrong with a program; a complex algorithm will be implemented as a complex program.

The authors suggest that the strongest evidence in favor of a specific meaning for the various elements of the difficulty equation is that for PL/S a high η_1 value indicates unstructured programming and a high N_2/η_2 value may be caused by unusually high occurrence of one or more of the six types of impurities that Halstead classified.

With respect to Halstead's equations for mental effort, language level, and potential volume (which they call information content), Christensen et al. say that experimental results are incomplete but interesting. Means for language level vary widely within a language, and "there is a suggestion that Language Level does not measure the language so much as it measures how the language is used in a program" (p. 385). Their cited λ values (p. 384) are:

Language	Mean λ	S.D.
PL/S	2.05	1.14
PL/I	1.53	0.92
370 assembler	0.91	0.79

Potential volume, V^*, not yet proven a "practical metric," is, if valid, a measure of how much function a program has--that is, its information content. For a series of eight programs implementing Euclid's algorithm and written in different languages, the mean V^* was 11.45, the variance 0.89, and the standard deviation 0.94 (p. 386).

In their conclusion, Christensen et al. stress how important it is to have measurement techniques for analyzing programs and designs. However, they also stress that errors in the "measurement instrument" will have to produce worthless results and that "strict and rigorous calibration" is required for any experiment (p. 386).

1.4 Warnier-Orr Diagrams

Around 1970 Warnier and his colleagues at Honeywell-Bull in Paris developed as a design tool diagrams of input and output data sets that resembled engineering parts explosion diagrams (Figure 1 is an example of an output report and indicates the hierarchical structure of a Warnier diagram.) Warnier (1974) later published a book on his design technique called Logical Construction of Programs, which Orr, working in the United States, used as the basis for his extended design technique,

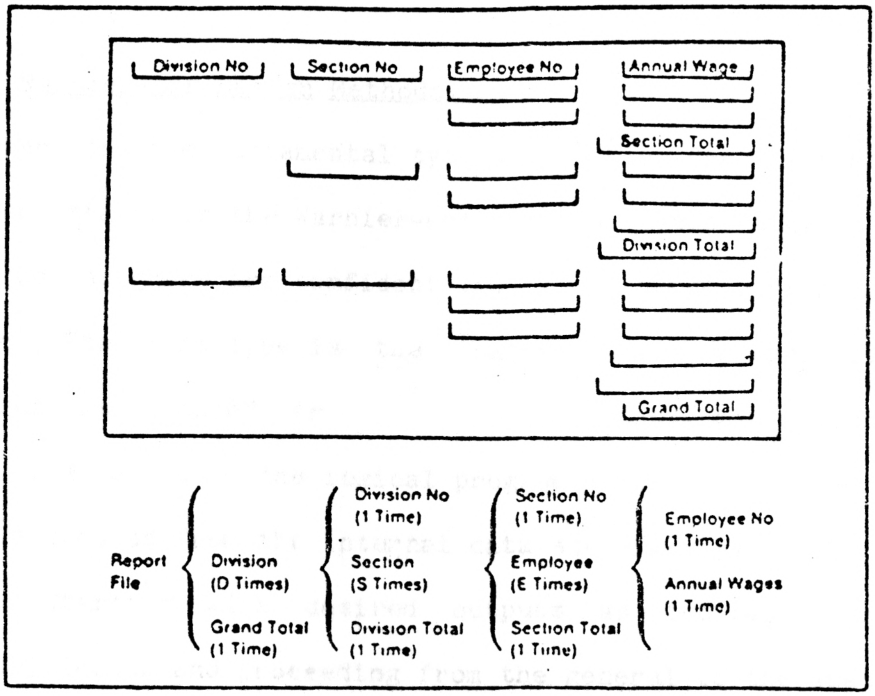

Figure 1
Higgins, 1979a, p. 2

called Warnier-Orr diagramming. Because Warnier-Orr diagrams are a practical tool for systems and data base design, they have become rather popular (Higgins, 1979b, p. 2).

1.4.1 Warnier-Orr Design Methodology

There are two fundamental types of Warnier-Orr diagrams produced at different stages in the Warnier-Orr design cycle, a cycle that repeats until the designers are confident that the program coding stage has been reached. The first type is the logical data structure diagram (Figure 2), which is deduced from the system requirements for the desired output; the second is the logical program structure diagram (Figure 3), which is deduced from the internal data structure needed to produce the output. Starting with desired outputs as the basis for finding the necessary inputs and proceeding from the general to the specific results in the cyclic construction of system flow.

The steps of the Warnier-Orr method--repeated from step 2 to step 9 until finished at some return to step 2--may be outlined as follows:

1. Discover the output requirements for the system as a whole.
2. Choose an undesigned part of the desired output.
3. Outline its system requirements.
4. Draw its logical data structure diagram.
5. Draw its preliminary logical program structure diagram.
6. Determine preliminary system flow.
7. Determine necessary input data for system flow.
8. Refine system flow.
9. Refine the logical program structure diagram.

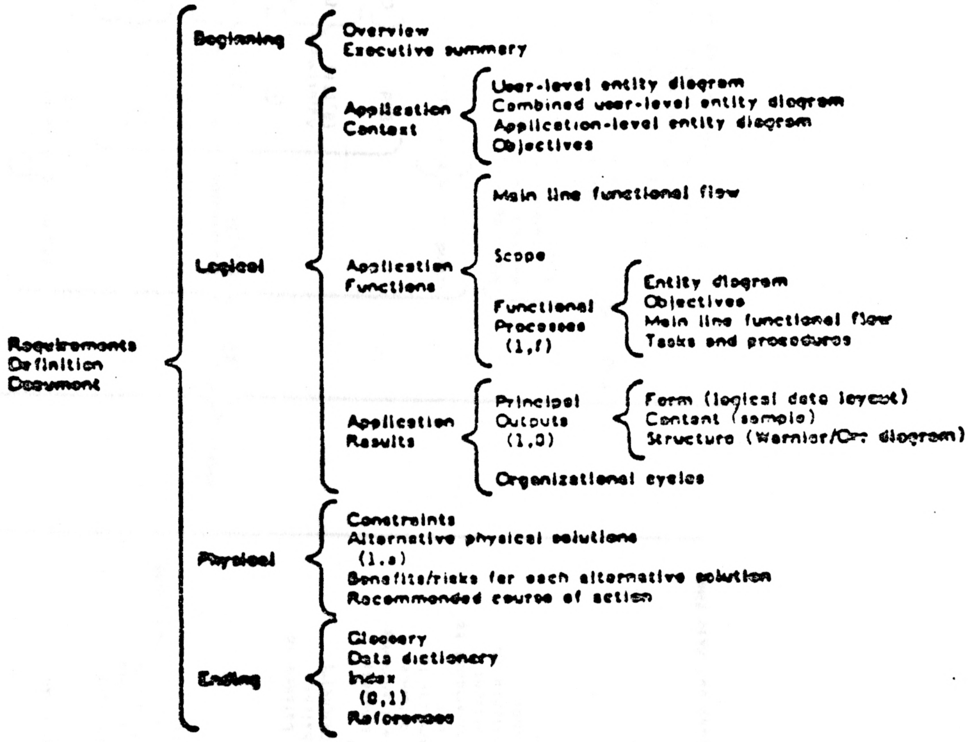

Figure 2
Higgins, 1979b, p. 191

- **ings count port**
  - **.Begin**
    - Print "SAVINGS ACCOUNT ACTIVITY"
    - Print "FOR THE YEAR ENDED";
    - Open inputs transaction file; account balance file; year end date file
    - Get year end date
    - Get first transaction record
  - **Year**
    - **.Begin**
      - Clear yearly deposit amount
      - Clear yearly withdrawal amount
      - Clear yearly current balance
      - Print year end date
      - Set end of accounts false
    - **Account (1,a)?7**
      - **.Begin**
        - Clear account deposit amount
        - Clear account withdrawal amount
        - Print "ACCOUNT #"; account number
        - Get account balance forward
        - Move account balance forward to monthly balance forward
        - Move account name to old account name
        - Set end of months false
      - **Month (12)?6**
        - ...See Part 2
      - **.End**
        - Move monthly current balance to account current balance
        - Print "TOTAL FOR ACCOUNT"; account number; account deposit amount; account withdrawal amount; account current balance
        - Add account deposit amount to yearly deposit amount
        - Add account withdrawal amount to yearly withdrawal amount
        - Add account current balance to yearly current balance
    - **.End**
      - Print "TOTALS FOR"; year number; yearly deposit amount; yearly withdrawal amount; yearly current balance
  - **.End**
    - Close transaction file; account balance file; year end date file

- **Month**
  - **.Begin**
    - Clear monthly deposit amount
    - Clear monthly withdrawal amount
    - Clear monthly current balance
    - Print month name; "ACTIVITY"; "BALANCE FORWARD OF"; balance forward amount "DATE";"ACTION";"DR";"CR";"BALANCE"
    - Move month name to old month name
    - Set end of days false
  - **Day (1,d)**
    - **.Begin**
      - Print date
      - Move transaction date to old transaction date
      - Set end of transaction to false
    - **Transaction (1,t)?4**
      - **Deposit (0,1)?1**
        - Add deposit amount to daily current balance
        - Print "DEPOSIT"; deposit amount
      - ⊕
      - **Withdrawal (0,1)?2**
        - Subtract withdrawal amount from daily current balance
        - Print "WITHDRAWAL"; withdrawal amount
      - ⊕
      - **Adjustment (0,1)?3**
        - Add adjustment amount to daily current balance
        - Print "ADJUSTMENT"; adjustment amount
      - **.End**
        - Get next transaction record
    - **.End**
      - Print daily current balance
  - **.End**
    - Move daily current balance to monthly current balance and balance forward amount
    - Print "FOR THE MONTH OF"; month name; monthly deposit amount; monthly withdrawal amount; monthly current balance
    - Add monthly deposit amount to account deposit amount
    - Add monthly withdrawal amount to account withdrawal amount

1.4.2 Theory of Warnier-Orr Structures

A Warnier-Orr diagram is laid out on the page using braces to show the expansion of a "universal" into its final "elements," which may be data elements or the Warnier-Orr process operators. Four basic structures corresponding to the concepts of structured programming make up the diagram (Higgins, 1979a, pp. 3-6) (see Figure 3 for examples of each):

1. Hierarchy structure--braces show successive decomposition of universals into elements.
2. Sequence structure--elements are listed sequentially within each hierarchical level.
3. Repetition structure--numbers or variables in parentheses beneath a universal indicate the range of repetition for a repeating subgroup. The structure (1,x) corresponds to a "do until" loop, (0,x) to a "do while" loop, and, say, (50) to a "do x = 1 to 50" loop.
4. Alternation structure--the repetition structure in the from (0,1) along with the exclusive or operator, + , represents alternative processes.

There are also two complex structures (not used in the present study because they are not implemented in the STRUCTURE(S) design package):

5. Concurrency structure--a + between two universals vertically shows concurrent operation.
6. Recursion structure--a broken brace following a universal name duplicating one to the left on the page indicates hierarchical repetition.

Four rules based on Warnier's programming theory determine the internal structure of the Warnier-Orr diagram for a program (Higgins, 1979a, p. 7):

1. The heirarchical structure of a program is deduced from the input data structure.

2. A repetitive input data structure produces a repetitive program structure.

3. An alternating input data structure produces an alternating program structure.

4. An alternating structure more than six levels deep must be determined from the output structure.

1.4.3 Present Usage of the Warnier-Orr Technique

The usefulness of Warnier-Orr diagrams to commercial custom-programming organizations is obvious: they are based on the needs of the user as outlined in a requirements document, they enforce data-driven structured programming, and they constitute an up-to-date record of the design cycle as they are being refined to the final stages. WO diagrams have not as yet been much used for designing other than business-type programs, although their potential usefulness in scientific applications and operating systems design is clear. If output requirements are well defined and system flow is complicated, WO diagrams will clarify and simplify the process of program design.

1.4.4 STRUCTURE(S): An Automated Warnier-Orr Diagram Drawing Package

It is easy to understand why Orr decided that a system to produce a Warnier-Orr diagram on a series of computer output pages and to list cross-references as well as remaining undefined references could be marketed successfully--the Warnier-Orr diagram for a program of substantial size quickly blossoms into a large, unwieldy sheet on which refinements and corrections are made with some difficulty and remaining unresolved segments may be overlooked.

The component of STRUCTURE(S) of interest in the present study is the "source input list," which is the user's input data that produces the Warnier-Orr diagram and reference lists. The input list phrases and tokens have, of course, a 1:1 relationship with the four Warnier-Orr structures and are suitable as input to the program written for this study that counts the operators and operands of a WO diagram.

All of the input lists used for designs in the present study may be found in the Appendix along with the program outputs. Following is a small segment from the input list for the Warnier-Orr diagram of the program that analyzes input lists; it shows the STRUCTURE(S) tokens:

```
COUNT;
 .BEGIN$;
 SETUP;
 SAVE DIAGRAM TITLE FOR OUTPUT TABLE$;
 SET HEAD OF LINKED LIST OF OPERANDS TO DIAGRAM TITLE$;
 SET HEAD OF LINKED LIST OF OPERATORS TO 'BRACE'$;
 FOR EVERY LINE  0-X;
 PRINT;
 .END$;
SETUP;
 .BEGIN$;
 CREATE LINKED LIST OF PREPOSITIONS/CONNECTIVES FROM INPUT FILES$;
 CREATE LINKED LIST OF INFINITIVE PHRASES FROM INPUT FILE$;
 .END$;
  .
  .
  .
FOR EVERY LINE;
 READ INPUT LINE$;
 FIRST CHAR = BLANK  0-1;
 +    FIRST CHAR = BLANK  0-1;
  .
  .
  .
```

The dollar sign is a terminal symbol to indicate that no brace occurs to the right of a phrase; therefore, absence of a "$" indicates that a brace is to be counted as present. The endline marker is obviously ";", the pair of parentheses around Warnier-Orr diagram repetition counts is represented by a "+". Sequence is indicated by the vertically arranged lists indented under headings which repeat the universal that the list is to appear within a brace.

These few tokens and the listed phrases are all that is needed to produce a Warnier-Orr diagram. Simple translation of the tokens as they are encountered in the input lists is all that must be done in order to count the actual Warnier-Orr process operators.

CHAPTER 2

AN OPERAND AND OPERATOR COUNTING TECHNIQUE FOR WARNIER-ORR DIAGRAMS

2.1 Experimental Assumptions

A basic assumption of the experiment which the rest of this paper will describe was that a Warnier-Orr program design diagram is composed of words and symbols that may be counted as operators and operands. As mentioned in Section 1.2, Halstead was sure that his software metrics were valid for technical English prose, and Kulm and Miller got good results for prose by counting "function words" as operators and "content words" as operands.

Since the design language of Warnier-Orr diagrams lies somewhere between technical prose and high-level programming languages with respect to "naturalness," there is little question that Halstead's software metrics should apply. The problem is to derive and justify an operator and operand counting technique. The approach taken in this experiment was the sample on of counting as operators the Warnier-Orr process operator symbols "{", "()", ",", and "+" along with the other logical operators (the arithmetic operators must be expressed in words, e.g., as "add" or "subtract"), verb phrases, prepositions, connectives, and the implied end-of-line marker, and as operands numbers and noun phrases.

That Warnier-Orr process operators and logical operators should be counted as Halstead operators is obvious. However, counting whole verb phrases and noun phrases rather than words as individual operators is a less refined technique than Kulm's and Miller's for prose. As briefly discussed at the end of Section 1.2, the assumption is that this relatively rough-grained approach is a suitable model of English prose structure as presently described by phrase structure grammars. Halstead noted that the operands and operators of English prose tend to alternate (see Section 1.2), and the importand implication of this fact is that operands--i.e., noun phrases, whose variations are endless--are positionally bracketed between operators--i.e., verb phrases, connectives, and punctuation symbols (possibly including an invisible end-of-line marker), whose variations may be conveniently limited in a design language. Therefore, it is reasonably rather than impossibly difficult to write a computer program to count the operators and operands of a Warnier-Orr diagram, and the arithmetic and logical operators furthermore seem to represent about the same degree of semiotic "complexity" as the linguistic "complexity" represent by simple word phrases--that is, what is signaled by a symbolic operator may be expressed in words by a verb.

The purpose of writing a counting program is, of course, to produce more consistent results than hand-counting would and to take advantage of the ready-made input that STRUCTURE(S) design language provides. Also, a practical complexity predictor for programs at the WO diagramming stage--if such is possible--would have to be automated.

2.2 Restriction on STRUCTURE(S)-Style Input Lists for Program COUNT

In order to simplify the parsing of STRUCTURE(S)-style input lists for program COUNT, a few restrictions were found to be necessary:

1. Simple phrase lines (i.e., those lines not representing the Warnier-Orr alternation or repetition structure) must be written in imperative voice, beginning with a one-word verb phrase, and be more than one word long.

2. All lines must be written in "telegraphic" stle, ilel, without articles. (Articles would be part of a noun phrase conted as one operand in any case.)

3. As much as convenient, the same noun phrase must be used repeatedly to describe repetitions of the same concept.

4. "Procedure names" must be one word long and appear as universals for the universals for the procedure elements at the first "call" in the design sequence.

5. A "procedure name" alone on a line with no following elements must be used to indicate subsequent repetitions of the sequence of lines it stands for.

6. Figures must always be used for numbers.

7. Except for figures and "$" or ";" or both; a simple phrase line must contain words only.

8. Except for single quote marks (with the conventional meaning), punctuation must not be used in phrase lines; separate phrase lines are used instead.

These few restrictions make the grammar of the STRUCTURE(S) input language determinate enough to be processible by a relatively simple program such as COUNT. That is, which of the four basic Warnier-Orr structures are represented in a line is determinable from the presence or absence of the relevant STRUCTURE(S) process operators "{" for hierarchy, "+" for alternation, " " without "+" for repetition, and none or "{" only for sequence. If a line is a simple sequence lines, then the first word must be an operator, the followingwords up to the first preposition (or the end of the line if there is none) constitute a noun phrase, operand, and the preposition is an operator or the first word of a two-word infinitive phrase operator, followed by a noun phrase operator, followed by a noun phrase up to the nxt preposition or the end of the line. A one-word line represents a procedure name operand.

2.3 Structure of Program COUNT

The listing for program COUNT appears at the end of the Appendix to this paper; the program is written in PL/I and makes extensive use of PL/I built-in string-processing functions. Input for COUNT, as described alrady, is the STRUCTURE(S) "source input list" for a WO diagram with the restrictions listed in Section 2.2. Output for COUNT, reproduced in the Appendix, consists of two tables--the first a list of the operators and η_1 and N_1 values for a WO diagram and the second a list of the operands and η_2 and N_2 values--along with the set of values for the nine Halstead metrics of inerest in this study--vocabulary (η), length (N), estimated most compact (potential) volume (V^*), language level (λ), mental effort (E), and time (T) in minutes.

Aside from the verb phrase - noun phrase alteration to the counting technique for prose, it initially seemed that Halstead's and Christensen's guidelines for counting program operators and operands could be followed closely for diagram operators and operands. However, it became apparent that a program procedure name, which is counted as an operator by Halstead, is not the same construction as its design representation in a WO diagram. In the program the procedure name represents a transfer of control from one location in the code to another; in the diagram the "procedure name" represents a subheading (noun phrase operand) paired with its brace (symbolic operator) to indicate the first occurrence of a named series of operations, and standing alone it represents subsequent occurrences ("procedure calls") of the named sequence. In this instance, for WO diagrams a natural language counting rule produces better internal consistency than Halstead's procedure call name rule for programming languages.

Otherwise, the diagram counting rules used in COUNT are straightforward implementations of the STRUCTURE(S)-to-WO-diagram transliterations described in Sections 1.4.4 and 2.1. Separate lined lists of operators and operands are constructed as encountered in the input, and ocurrence counts are basic structures of a WO diagram (hierarchy, sequence, repetition, and alteration) are represented in an input line is easily determined by searching for the corresponding STRUCTURE(2) symbols (see Figure 4) for diagrammatic explanations): a brace (hierarchy) is logged for each line without a "$", a "()" pair and a "," plus the other particular operands and operators (repetition) are logged for each line without a "*" and a "-" are encountederd, and logical operators, and the WO standard operators ".begin", ".end", and ".skip" are logged as found. A single word, other than one of the standard operators, appearing alone on a line must be a "perocedure name" and is logged in the oprands list on each such occurrence as well as on any mention in a simple phrase line. Simple phrase lines--representing the WO sequence structure--are distinguished by a lack of the symbols indicating a repetition or alteration structure. They are parsed one word at a time based on the rule that the first word in each such line must be an operator (verb). The line is searched for a preposition or connective by comparing each successive word to an already set up linked list of the prepositions and connectives most likely to appear in a WO diagram. If none is found, the unprocessed part of the input line is printed on the terminal screen for the program user to signal interactively how processing is to be completed. If no prepositions or connective unknown to COUNT appears in the line portion displayed on the screen, the user signals that portion is to be logged

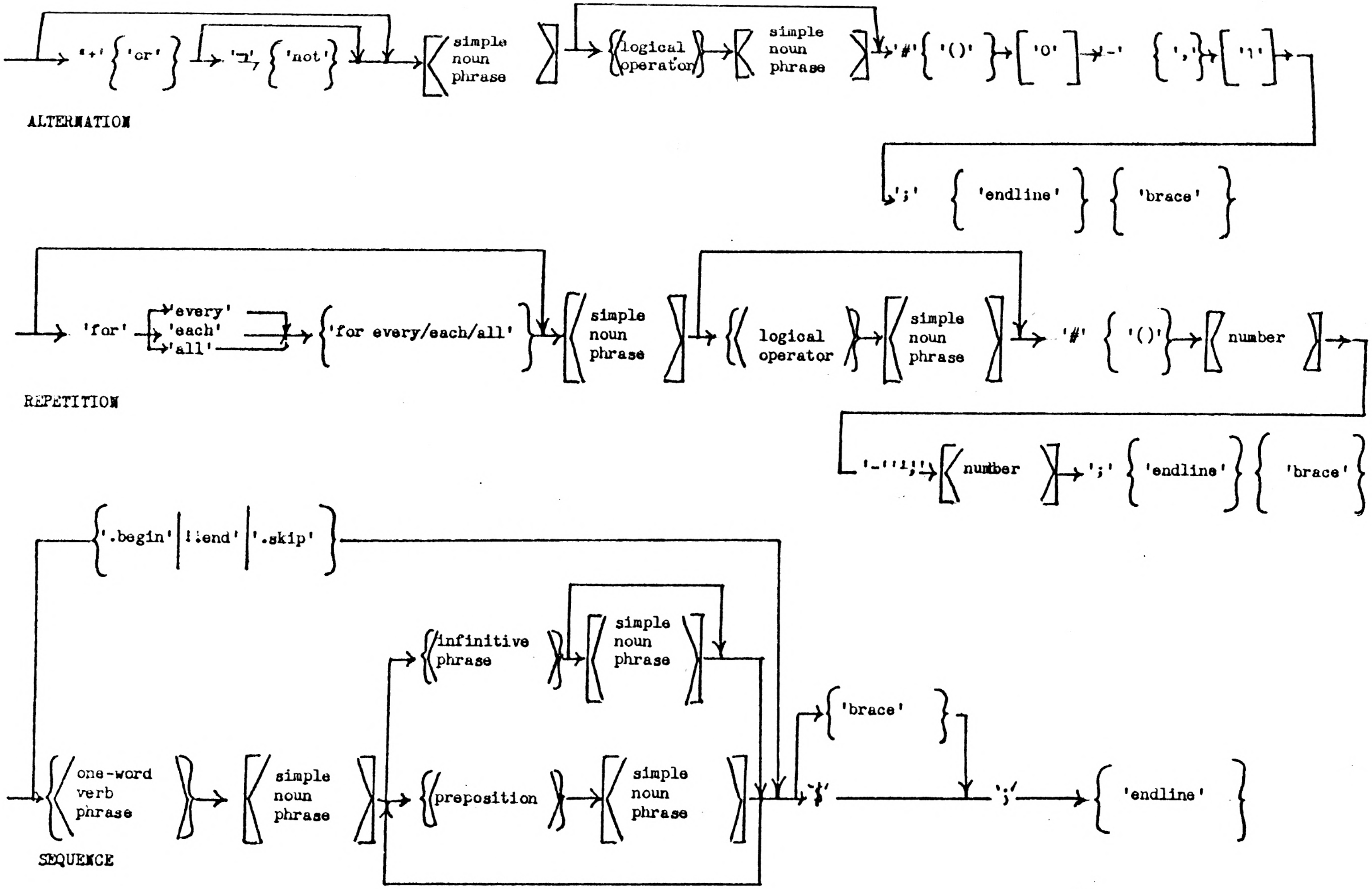

Figure 4

Syntex diagrams for Warnier-Orr basic structure as implemented in this study using STRUCTURE(S) source input lists. Literals are in single quotes, variable tokens are in angular brackets. The Warnier-Orr diagram operators and operands logged are shown within braces and square brackets, respectively. Where the Warnier-Orr operator or operand logged differs from the STRUCTURE(S) input token, the former follows the latter immediately.

as an operand (noun phrase) to the end and processing continues with the next input line. If an unknown preposition or connective appears in the problematic line portion, the user writes the word on the screen, and COUNT logs it and any preceding noun phrase in the operators and operands lists; processing of the line continues until the end is detected. The presence of the word "to" in a phrase line constitutes a particular problem; when found, it and the next word are compared to an already set up linked list of infinitive phrases most likely to be found in a WO diagram. Again, if no match is found the program user must signal whether the occurrence of "to" represents a preposition followed by its object (an operator followed by an operand) or of an infinitive phrase (a two-word opeator).

When the end of the input file has been reached, COUNT prints the tables or operators and operands and computes and prints the values of the nine Halstead metrics.

2.4 Program Operator and Operand Counting Programs

As already mentioned, a small program to list the tokens in each program source code listing was written that simply constructs an output file of each token along with the number of occurrences. These are combined by hand into a master list of program operators and operands usng Halstead's definitions with Christensen et al.'s clarifications. The master list provides values for the program η_1, N_1, η_2, and these are the input for another small program that is basically the same as the code in COUNT that computes the nine Halstead values for a diagram. The Halstead metric outputs for the nine programs analyzed in this experiment appear in the Appendix after those for their corresponding diagrams.

CHAPTER 3

PREDICTIVE POWER OF HALSTEAD DESIGN VALUES FOR PROGRAM VALUES

3.1 Experimental Hypothesis

As mentioned in Section 1.1, the purpose of this experiment was to find whether Halstead complexity values are derivable for WO diagrams and, if so, whether theses values can be used to predict the complexity of programs written from the diagrams.

It was hypothesized that Halstead metrics for WO diagrams should be internally consitent and that the values should have a fairly consistent relationship to the same values for programs. This, of course, is not to imply that the set of operators and operands for a WO diagram maps directly into the set of operators and operands for the program written from the diagram. While there is considerable overlap between the symbolic operators used for WO diagrams and for a high-level programming language, there are also several special-purpose operators for WO diagrams that do not translate straightforwardly into program operators (e.g., the hierarchical brace) and programming languages use many arithmetic, logical, and special-purpose operators not required by a design language. Furthermore, the alteration structure of a Warnier-Orr diagram (see Figure 3) is a quite different construct from the "IF THEN ELSE" of a high-level programming language, and with respect to the WO sequence structure a noun phrase in a WO diagram can only rarely be translated into a single program variable name or a verb phrase (other than "add", "subtract", "multiply", or "divide" into a single arithmetic or logical operator. The hypothesized fairly consistent relationship between Halstead values for diagrams and programs must be based on some consistency of their "deep structure" (a term used by phrase structure grammarians with reference to the still poorly understood psychology of language).

3.2 Experimental Procedure

Six WO diagrams for different program designs were prepared; five were worked up to the program coding point and one was left at a fairly abstract level for comparison. Two of the designs, BKB2PFGP and BKB2PIRW, were for modules that became part of a diskette file management system being considered for a small operating system. These designs and their resulting programs were subjected to an inspection and review process, and the programs were approximately 120 and 60 lines long, respectively. The largest design--for program COUNT, about 650 lines long--was the WO diagram for the program that counts operators and operands in WO diagrams; that is, this experiment's counting program design was used as input for the program it produced. Diagram LINKED LIST was for a demonstration program of modest length--about 300 lines--that produces and manipulates several singly linked lists. Diagrams SORT1 and SORT2 were for short programs (about 15 and 25 lines, respectively) to implement a bubble sort, the former for a fixed-length array of elements and the latter for a doubly linked list of undefined length. Diagram SORT1 was prepared to the coding point; diagram SORT2 was left at a preliminary high level of design and the program coded without a detailed design.

Programs BKB2PFGP, BKB2PIRW, and SORT1 were written in PLDS, a subset of PL/I used for systems programming; COUNT, LINKED LIST, and

SORT1 were written in PL/I. In addition, UC assembler language programs were produced for BKB2PFGP, BKB2PIRW, and SORT1; these programs were 440, 400, and 90 lines long, respectively.

The six diagrams were translated into STRUCTURE(S) - style lists for input to program COUNT. The nine programs were compiled or assembled and run, after which the source file for each was used as input to a small token-counting program whose output was used to count program operators and small program essentially the same as procedure PRINT in program COUNT. Program operators and operands were hand-counted because the counting method is simple and well defined (Halstead's definitions described in Section 1.2 with Christensen et al.'s clarifications described in Section 1.3.2 were followed) and because writing a program to do the counting would have required a good deal of time and was not of particular relevance to this study.

3.3 Results

The output tables and lists of Halstead values for diagrams and programs compose the raw data used to investigate the hypothesis that a program's complexity may be estimated from the complexity of the Warnier-Orr diagram used to design it.

3.3.1 Validity of the Diagram Operator and Operand Counting Technique

An important indicator of whether in fact the hypothesis of this experiment can be tested is some sign that Halstead's metrics have been successfully adapted to the analysis of Warnier-Orr diagrams--that is, whether program COUNT meets Christensen et al.'s standard as a well-calibrated measurement instrument.

Two ways of checking COUNT's calibration are available: comparison the published language level (λ) values for natural and high-level programming languages with those for the WO diagrams, and comparison of Halstead's correlation coefficient for length (N) and estimated length (est. N) of a large sample of programs with the correlation coefficient for diagram N and est. N values (see Appendix for explanation of the correlation computations).

As listed in Section 1.3.1, Fitzsimmons and Love's cited λ for English prose is 2.16 + 0.86 and their λ for PL/I is 1.53 + 0.96, almost the same as that cited by Christensen et al. The obtained in this experiment for five WO diagrams (excluding the SORT2 diagram, which was intentionally left uncompleted) is 1.18 + 0.44. This value is within one standard deviation of the mean for both English prose and PL/I, which is acceptable although one would prefer to have the mean value for diagrams between the two others rather than below them. The large standard deviations preclude using relative λ s to reach a strong conclusion in any case. Christensen et al.'s λ for 370 assembler language is 0.91 + 0.79, which is within one standard deviation of that for PL/I.

What other investigators emphasize about values is that they tend to increase from low-level to high-level languages. The diagram and program values for this experiment are as follows:

Language	Mean λ	S.D.
PL/I, PLDS	1.35	0.46
Diagram	1.18	0.47
UC assembler	0.44	0.05

The implication is that diagram language may be somewhat more restricted than the high-level programming languages but both are approximately the same and of distinctly higher level than the assembler language. As Christensen et al. point out, may be more of an indication of how a language has been used in a particular application than of the language's inherent "level" (see Section 1.3.2).

A better indication of the internal consistency of the Halstead values for diagrams is the correlation between N (equation 2) and est. N (equation 7). As cited previously, Halstead found a correlation coefficient of 0.98 between N and est. N for a large series of programs. The diagram correlation coefficient for N and est. N in the present experiment is 0.95, which is significant at the 1 percent level for a sample size of 6. With the COUNT program excluded because high usage of PL/I string-processing functions confounded the est. N value, the correlation coefficient for the programs of this experiment is 0.96, and that for programs and designs combined is also 0.96. These values used in this experiment for diagrams and programs separately and combined do meet Halsted's criteria. They tend to strengthen the assumption that further conclusions may be drawn about relationships between diagram and program values for the other Halstead metrics.

3.3.2 Diagram:Program Ratios of the Halstead Metrics

To determine what the "fairly consistent" relationship between diagram values and program values is, diagram:program ratios were calculated for the Halstead metrics of this experiment. Table 3.1 lists the Halsted values for estimated length and actual length for all diagrams and programs along with the diagram:program length ratios. Tables 3.2 through 3.5 give the Halstead values and diagram:program ratios for the other metrics.

TABLE 3.1
Relationship of Estimated Length to Actual Length for Diagrams and Programs

Title	Actual length (N)	Estimated length (est. N)	Est. N error (%)	Diagram: program ratio (N:N)
BKB2PFGP				
diagram	352	314.0	-11	
PLDS	542	541.1	-17	65
assembler	957	740.0	-23	37
BKB2PIRRW				
diagram	182	238.6	31	
PLDS	285	292.6	3	64
assembler	398	454.6	14	46
LINKED LIST				
diagram	396	353.3	-11	
PL/I	508	382.9	-25	78
COUNT				
diagram	580	551.5	- 5	
PL/I	1845	233.1	-87	31
SORT1				
diagram	76	128.8	69	
PLDS	87	76.2	-12	88
assembler	159	232.7	46	47
SORT2				
diagram	89	150.8	70	
PL/I	222	155.8	-30	70

TABLE 3.2
Volume and Volume Ratios for Diagrams and Programs

Title	Volume (V)	Diagram:program ratio (V:V)
BKB2PFGP		
diagram	2095.9	
PLDS	3445.8	61
assembler	6644.0	32
BKB2PIRW		
diagram	1032.4	
PLDS	1676.6	62
assembler	2544.1	41
LINKED LIST		
diagram	2410.6	
PL/I	3134.3	77
COUNT		
diagram	3827.9	
PL/I	10412.9	37
SORT1		
diagram	380.0	
PLDS	388.0	97
assembler	897.4	42
SORT2		
diagram	460.1	
PL/I	1156.5	40

TABLE 3.3
Language Level and Language Level Ratios
for Diagrams and Programs

Title	Language Level (gamma)	Diagram:program ratio (gamma:gamma)
BKB2PFGP		
diagram	1.76	
PLDS	1.79	0.98
assembler	0.51	3.45
BKBWPIRW		
diagram	0.96	
PLDS	1.55	0.62
assembler	0.38	2.53
LINKED LIST		
diagram	1.63	
PL/I	2.86	0.57
COUNT		
diagram	0.93	
PL/I	0.05	18.6
SORT1		
diagram	0.62	
PLDS	0.71	0.87
assembler	0.43	1.44
SORT2		
diagram	0.59	
PL/I	0.62	0.95

TABLE 3.4
Estimated Abstraction Level, Difficulty, Structure, and Abstraction Level Ratios for Diagrams and Programs

Title	Estimated abstraction level (est. L)	Difficulty* (N2/ $_2$)	Structure* (η_1)	Diagram:program ratio (est. L:est. L)
BKB2PFGP				
diagram	0.0290	3.63	19	
PLDS	0.0228	3.55	22	1.27
assembler	0.0088	4.58	43	3.29
BKB2PIRW				
diagram	0.0304	2.86	23	
PLDS	0.0304	3.12	20	1.00
assembler	0.0122	4.06	35	2.49
LINKED LIST				
diagram	0.0260	3.65	21	
PL/I	0.0302	2.98	20	0.86
COUNT				
diagram	0.0156	4.02	21	
PL/I	0.0022	7.79	28	7.09
SORT1				
diagram	0.0402	2.62	19	
PLDS	0.0427	3.90	12	0.94
assembler	0.0218	3.27	28	1.84
SORT2				
diagram	0.0357	2.67	21	
PL/I	0.0231	2.85	17	1.55

*According to Christensen et al. (1981).

TABLE 3.5
Mental Effort, Time, and Mental Effort Ratios for Diagrams and Programs

Title	Mental effort* (E)	time (T min)	Diagram:program ratio (E:E)
BKB2PFGP			
diagram	72246.7	66.9	
PLDS	151264.0	140.1	0.48
assembler	758445.9	702.3	0.09
BKB2PIRW			
diagram	33926.4	31.4	
PLDS	55149.8	51.1	0.62
assembler	208195.0	192.8	0.16
LINKED LIST			
diagram	92645.5	85.8	
PL/I	103785.5	96.1	0.89
COUNT			
diagram	246012.2	227.8	
PL/I	4733143.0	4382.5	0.05
SORT1			
diagram	9443.3	8.7	
PLDS	9094.5	8.4	1.04
assembler	41201.7	38.1	0.23
SORT2			
diagram	12885.0	11.9	
PL/I	50043.2	46.3	0.26

*Called "information content" by Christensen et al. (1981).

In calculating the means and standard deviations of the diagram:program ratios, it was decided that COUNT and SORT2 should be excluded because program values are distorted for the former by PL/I string-processing functions, and program and diagram values differ greatly for the latter, whose design was intentionally left at an abstract level to demonstrate that such would be the case. Excluding COUNT, SORT2 has the highest diagram:program ratio for estimated abstraction level of all the high-level-language diagrams (Table 3.4). Program COUNT has the highest "difficulty" value of all diagrams and programs--almost twice that of its diagram and considerably higher than the difficulty values of the assembler-language programs--but its "structure" value is not overly high, which is proper for a structured program (Table 3.4). Therefore, the "poor" diagram:program results for SORT2 and COUNT seem intuitively reasonable.

Means and standard deviations of the diagram:program ratios for which a statistically significant (or nearly so) relationship exists between the diagram and program values are listed below.

Those for length are:

Diagram:assembler	43.3 + 4.5
Diagram:high-level	73.8 + 9.9

Those for volume are:

Diagram:assembler	41.0 + 2.2
Diagram-high-level	74.3 + 14.6

Those for estimated abstraction level are:

Diagram:assembler	2.54 + 0.59
Diagram:high-level	1.02 + 0.15

Correlation coefficients are 0.98 for length, 0.99 for volume, and 0.84 for estimated abstraction level. For a sample size of 4, which is the number of high-level-language programs in this experiment, a correlation coeffcient of 0.95 or above is significant at the 5 percent level. Only length and volume exceed this requirement, but because of the small sample size a significant relationship cannot be excluded for estimated abstraction level. Correlation coefficients could not be calculated for assembler-language programs because of small sample size (n = 3), but standard deviations of the mean for length and volume are relatively smaller for assembly-language programs, which is a good sign that a significant relationship between diagram values and program values could be shown in a larger study.

The correlation coefficient for diagram line counts and high-level-language program lines of code is 0.90 (n = 4), which fails significance at the 5 percent level although it is somewhat higher than the correlation coefficent for estimated abstraction level. This may be an indication that Halstead's length and volume metrics are rather more fundamental measures of program (and WO diagram) size than is the lines of code measure.

Although significant relationships between diagram and program values for Halstead's estimated length, language level, mental effort, and time are not indicated--perhaps because they are more vaguely conceived ideas--the values by themselves are of some interest. The mental effort and time values seem to indicate that a WO diagram requires about half as much work as its high-level-language program and that an assembler-language program is 3 or 4 times harder to write than a high-level-language one. With respect to the language level (λ) results, the overlapping values for diagrams and high-level language are at least reasonable compared with the results of others, as already discussed.

3.3.3 The V^* metric

According to Halstead, potential volume (V^*) is a language-independent representation of the minimum size of a program and therefore should be approximately constant for versions of a program written in different languages.

This experiment did not produce constant V^* values for the six diagrams and nine programs. V^* values are lower for WO diagrams than for high-level-language programs, and assembler-language programs have the lowest V^* values. Means and standard deviations of diagrams and programs combined are listed below.

BKB2PFGP	65.8 + 9.0
BKB2PIRW	37.8 + 9.3
COUNT	51.3 + 18.4
LINKED LIST	78.7 + 16.0
SORT1	17.1 + 1.8
SORT2	21.6 + 5.2

Even assuming that the figures for COUNT and SORT2 are worthless for computing V^*, these results compare rather unfavorably with Christensen et al.'s previously cited $V^* = 11.45 + 0.94$ for eight implementations of Euclid's algorithm where the standard deviation was somewhat less than 10 percent of the mean. The relatively small size of this experiment (two or three versions of each program concept) may be one cause of poor results for V^*.

3.4 Conclusions

The hypothesis of this experiment--that there should be a fairly consistent relationship between Halstead values for WO diagrams and for the programs written from them--is borne out, with some reservations because of the small size of this experiment, for the Halstead metrics length (N) and volume (V) and possibly also estimated abstraction level (est. L). Results for estimated length, language level, most compact volume, and the time are inconclusive. Fortunately, length and volume, based on the vocabulary of operators and operands in a program (or WO diagram) rather than the conventional "lines of code" size measurement, are the strongest and apparently most accepted of Halstead's metrics (Christensen et al., 1981, pp. 377-378). If the results of a much larger study were to bear out those of this small preliminary one, then masurements of a WO diagram's length and volume might easily be calculated from the STRUCTURE(S) source input list or some other diagram adaptation to serve as a predictor of program length and volume.

A study of correlations between program Halstead values and diagram Halstead values produced by a finer-grained operator and operand counting program would also be of interest. Kulm's and Miller's techniques for counting operators and operands in technical English prose are far more involved than the simple verb phrase and noun phrase scheme used here for WO diagrams, but there is some indication that the simple method is accurate for the short phrases of a WO diagram and that a counting method which separately considered adjectives, adverbs, articles, and other grammatical constructions for diagrams and programs as a result of relatively higher operator counts for diagrams.

Aside from the large questions of whether Halstead's metrics do tap some fundamental "complexity" represented by a linguistic expression and whether knowldege of the "complexity" of a computer program is useful in engineering better software, some doubt will remain as to the accuracy of this experiment unless its results are independently corroborated. A preliminary study can do little more than be interesting and help to direct future study. Other investigations of Halstead's theories all seem to be preliminary in nature, and it is unclear whether some or all of his metrics will one day be of practical use in software engineering. If so, and if the Warnier-Orr diagramming technique continues to prosper, the two approaches are apparently candidates for combination into a refined design methodology.

REFERENCES

Bandyopadhyay, S. K. (1981a). A study on program level dependency of implemented algorithms on its potential operands. Bangalore, India: Aeronatuical Development Establishment, SIGPLAN Notices, February, pp. 18-25.

Bandyopadhyay, S. K. (1981b). Theoretical relationships between potential operands and basic measureable properties of algorithm structure. Bangalore, India:Aeronautical Development Establishment. SIGPLAN Notices, February, pp. 26-34.

Chapman, D. G., and Schaufele, R. A. (1970). Elementary Probability Models and Statistical Inference. Waltham, Mass.:Xerox College Publishing, pp. 238-250, 337.

Christensen, K., Fitsos, G. P., and Smith, C. P. (1981). A perspective on software science. IBM Systems Journal 20 (4):372-387.

Fitzsimmons, A., and Love, T. (1978). A review and evaluation of software science. Computing Surveys 10(1):3-18.

Halstead, M. H. (1977). Elements of Software Science. New York:Elsevier.

Halstead, M. H. (1979a). Advances in software science. Advances in Computers 18:119-172.

Halstead, M. H. (1979b). Software science. Encyclopedia of Computer Science and Technology (J. Belzier, ed.). New York:Dekker, Vol. 13, pp. 242-262.

Higgins, D. A. (1979a) Warnier-Orr diagrams. Computer Programming Management. New York:Auerbach, pp. 1-8.

Higgins, D. A. (1979b). Program Design and Construction. Englewood Cliffs, N. H.:Prentice-Hall.

Kulm, G. (1975). Language level applied to the information content of technical prose. Collective Phenomena and the Applications of Physics to Other Fields of Science. (N. A. Chigier and E. A. Sterns, eds.). Fayetteville, N. Y.:Brain Research Publications, pp. 401-408.

Langston Kitch & Associates (1978). Langston Kitch STRUCTURE(S). Topeka, Ks.:Advanced Systems, pp. 1-21.

Miller, G. A., Newman, E. B., and Friedman, E. A. (1958). Length frequency statistics of written English. _Information Contributions_. 1:370-389.

Stroud, J. M. (1966). The fine structure of psychological time. _Annals of the New York Academy of Sciences_, pp. 623-631.

Warnier, J. D. (1974). _Logical construction of Programs_. New York:Van Nostrand Reinhold.

APPENDIX A

Calculation of
Correlation Coefficients

Calculation of Correlation Coefficients for Diagram and Program Halstead Values

Correlation coefficients for diagram and program Halstead values were computed from the formula given by Chapman and Schaufele (1970, p. 248):

$$S_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (X_1 -- X)(y_1 -- y)$$

$$S^2{}_x = \frac{1}{n-1} \sum_{i=1}^{u} (X_1 -- X)^2$$

where
n = total number of observations
X_i = ith x value
y_i = ith y value

The sample correlation coefficient is

$$r = \frac{S_{XY}}{S_X S_Y}$$

It is assumed for the purpose of computing r values that diagram and program values are jointly normaly distributed so if p = 0 the implication is that the two data sets are independent. Therefore, low r values suggest that diagram values and program values are not related. Rejection of the null hypothesis H:p = 0 because of high r values implies that diagram and program values are dependent.

Table A2.7 of Chapman and Schaufele (1970, p. 337) gives the critise levels for the distribution of r. Those of interest here are:

Sample Size	5 percent Significance Level	1 percent Significance Level
4	0.950	0.990
6	0.811	0.917

In calculating correlation coefficients, diagram Halstead values were assumed to be the independent variable X and program Halstead values to be the dependent variable Y. Following are the X and Y values for N, V, and est. L.

Length (N)

X	396	76	352	182
Y	508	87	542	285

Volume (V)

X	2410.6	380.0	2095.9	1032.4
Y	3134.3	388.0	3445.8	1676.6

Estimated abstraction level (est. L)

X	0.0260	0.0402	0.0290	0.0304
Y	0.0302	0.0427	0.0228	0.0304

The computed r values are listed in the Results section of the text.

APPENDIX B

Data for program LINKED LIST

HALSTEAD'S COMPLEXITY MEASURES FOR DIAGRAM LINKED LIST

VOCABULARY = ETA = ETA-1 + ETA-2 = 68

LENGTH = N = N1 + N2 = 396

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 353.3

VOLUME = V = N LOG2 ETA = 2410.6

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1)(ETA-2/N2) = 0.0260

MOST COMPACT VOLUME = V* = LV = 62.7

LANGUAGE LEVEL = GAMMA = (L**2) * V = 1.63

MENTAL EFFORT = E = V/L = 92645.5

TIME (IN MINUTES) = T = E / (S * 60) = 85.8

HALSTEAD'S COMPLEXITY MEASURES FOR LLIST PLI PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 72

LENGTH = N = N1 + N2 = 508

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 382.9

VOLUME = V = N LOG2 ETA = 3134.3

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0302

MOST COMPACT VOLUME = V* = LV = 94.7

LANGUAGE LEVEL = GAMMA = (L**2) * V = 2.86

MENTAL EFFORT = E = V/L = 103785.5

TIME (IN MINUTES) = T = E / (S * 1080) 96.1

TABLE 1. OPERATORS OF DIAGRAM LINKED LIST

		OPERATOR	COUNT
	1	BRACE	39
	2	.BEGIN	7
	3	.END	7
	4	DISPLAY	13
	5	SET	13
	6	TO	15
	7	ASK	6
	8	TO ENTER	6
	9	GET	6
	10	=	22
	11	()	27
	12	,	27
	13	OR	14
	14	NOT	10
	15	.SKIP	1
	16	FOR EVERY/EACH/ALL	3
	17	ALLOCATE	1
	18	FOR	2
	19	ASSIGN	2
	20	>	2
ETA-1 =	21	FREE	1
			224 = N1

TABLE 2. OPERANDS OF DIAGRAM LINKED LIST

	OPERAND	COUNT
1	DIAGRAM LINKED LIST	1
2	PROCEDURE OPCODES	2
3	OPCODES LIST	1
4	HEAD POINTER	16
5	NULL	9
6	OPCODE	11
7	'QUIT'	3
8	0	27
9	1	24
10	X	1
11	'INSERT'	1
12	'LOCATE'	1
13	'DELETE'	1
14	'PRINT'	1
15	'OPCODES'	1
16	NEW ELEMENT KEY	4
17	NEW ELEMENT DATA	2
18	PROCEDURE INSERT	1
19	LOCATE ELEMENT KEY	2
20	PROCEDURE LOCATE	1
21	DELETE ELEMENT KEY	2
22	PROCEDURE DELETE	1
23	PROCEDURE PRINT	1
24	ERRORMESSAGE	6
25	SPACE	1
26	NEW ELEMENT NODE	3
27	PRESENT POINTER	4
28	NEW ELEMENT POINTER	2
29	ELEMENT KEY	1
30	ELEMENT DATA	1
31	FUNCTION COMPLETE MESSAGE	4
32	PROCEDURE FIND	3
33	DUPLICATE KEY	7
34	'YES'	2
35	'NO'	5
36	ELEMENT NODE	2
37	E	2
38	LAG POINTER	1
39	HEAD ELEMENT KEY	2
40	LAG POINTER LINK	3
41	PRESENT ELEMENT LINK	2
42	SPACE ALLOCATED	1
43	PRESENT ELEMENT	1
44	PRESENT ELEMENT DATA	1
45	PRESENT ELEMENT KEY	1

```
          46     NEW ELEMENT LINK                                  2
EPA-2 =   47     NEW ELEMENT                                       1
  --------------------------------                          ----------
                                                            172 = N2
```

```
LINKED LIST;
 .BEGIN$;
 OPCODES;
 .END;
OPCODES;
 .BEGIN$;
 DISPLAY OPCODES LIST$;
 .END$;
LINKED LIST.END;
 SET HEAD POINTER TO NULL$;
 ASK USER TO ENTER OPCODE$;
 GET OPCODE$;
 OPCODE = 'QUIT' #0-1;
 + ¬ OPCODE = 'QUIT' #0-1;
OPCODE = 'QUIT';
 .SKIP$;
¬ OPCODE = 'QUIT';
 FOR EVERY OPCODE #0-X;
FOR EVERY OPCODE;
 OPCODE = 'INSERT' #0-1;
 + OPCODE = 'LOCATE' #0-1;
 + OPCODE = 'DELETE' #0-1;
 + OPCODE = 'PRINT' #0-1;
 + OPCODE = 'OPCODES' #0-1;
 + ¬ OPCODE = 'QUIT' #0-1;
 ASK USER TO ENTER OPCODE$;
 GET OPCODE$;
OPCODE = 'INSERT';
 ASK USER TO ENTER NEW ELEMENT KEY$;
 GET NEW ELEMENT KEY$;
 ASK USER TO ENTER NEW ELEMENT DATA$;
 GET NEW ELEMENT DATA$;
 INSERT;
OPCODE = 'LOCATE';
 ASK USER TO ENTER LOCATE ELEMENT KEY$;
 GET LOCATE ELEMENT KEY$;
 LOCATE;
OPCODE = 'DELETE';
 ASK USER TO ENTER DELETE ELEMENT KEY$;
 GET DELETE ELEMENT KEY$;
 DELETE;
OPCODE = 'PRINT';
 PRINT;
OPCODE = 'OPCODES';
 OPCODES;
¬ OPCODE = 'QUIT';
 DISPLAY ERRORMESSAGE$;
INSERT;
 .BEGIN$;
 ALLOCATE SPACE FOR NEW ELEMENT NODE$;
 HEAD POINTER = NULL #0-1;
 + ¬ HEAD POINTER = NULL #0-1;
 .END$;
LOCATE;
 .BEGIN$;
```

```
 HEAD POINTER = NULL #0-1;
 + ¬ HEAD POINTER = NULL #0-1;
 .END$;
DELETE;
 .BEGIN$;
 SET PRESENT POINTER TO HEAD POINTER$;
 HEAD POINTER = NULL #0-1;
 + ¬ HEAD POINTER = NULL #0-1;
 .END$;
PRINT;
 .BEGIN$;
 SET PRESENT POINTER TO HEAD POINTER$;
 HEAD POINTER = NULL #0-1;
 + ¬ HEAD POINTER = NULL #0-1;
 .END$;
INSERT.HEAD POINTER = NULL;
 SET HEAD POINTER TO NEW ELEMENT POINTER$;
 ASSIGN ELEMENT KEY TO NEW ELEMENT NODE$;
 ASSIGN ELEMENT DATA TO NEW ELEMENT NODE$;
 DISPLAY FUNCTION COMPLETE MESSAGE$;
INSERT.¬HEAD POINTER = NULL;
 FIND;
 DUPLICATE KEY = 'YES' #0-1;
 + ¬ DUPLICATE KEY = 'YES' #0-1;
LOCATE.HEAD POINTER = NULL;
 DISPLAY ERRORMESSAGE$;
LOCATE.¬HEAD POINTER = NULL;
 FIND;
 DUPLICATE KEY = 'NO' #0-1;
 + ¬ DUPLICATE KEY = 'NO' #0-1;
DELETE.HEAD POINTER = NULL;
 DISPLAY ERRORMESSAGE;
DELETE.¬HEAD POINTER = NULL;
 FIND;
 DUPLICATE KEY = 'NO' #0-1;
 + ¬ DUPLICATE KEY = 'NO' #0-1;
PRINT.HEAD POINTER = NULL;
 DISPLAY ERRORMESSAGE$;
PRINT.¬HEAD POINTER = NULL;
 FOR EVERY ELEMENT NODE #0-E;
FIND;
 .BEGIN$;
 SET DUPLICATE KEY TO 'NO'$;
 SET PRESENT POINTER TO HEAD POINTER$;
 SET LAG POINTER TO HEAD POINTER$;
 FOR EVERY ELEMENT NODE #0-E;
 .END$;
INSERT.¬HEAD POINTER = NULL.DUPLICATE KEY = 'YES';
 DISPLAY ERRORMESSAGE$;
INSERT.¬HEAD POINTER = NULL.¬DUPLICATE KEY = 'YES';
 NEW ELEMENT KEY > HEAD ELEMENT KEY #0-1;
 + ¬ NEW ELEMENT KEY > HEAD ELEMENT KEY #0-1;
LOCATE.¬HEAD POINTER = NULL.DUPLICATE KEY = 'NO';
 DISPLAY ERRORMESSAGE$;
LOCATE.¬HEAD POINTER = NULL.¬DUPLICATE KEY = 'NO';
```

```
 SET LAG POINTER LINK TO PRESENT ELEMENT LINK$;
 FREE SPACE ALLOCATED FOR PRESENT ELEMENT$;
 DISPLAY FUNCTION COMPLETE MESSAGE$;
FOR EVERY ELEMENT NODE;
 DISPLAY PRESENT ELEMENT DATA$;
 DISPLAY PRESENT ELEMENT KEY$;
 SET PRESENT POINTER TO PRESENT ELEMENT LINK$;
NEW ELEMENT KEY > HEAD ELEMENT KEY;
 SET NEW ELEMENT LINK TO LAG POINTER LINK$;
 SET LAG POINTER LINK TO NEW ELEMENT POINTER$;
 DISPLAY FUNCTION COMPLETE MESSAGE$;
¬ NEW ELEMENT KEY > HEAD ELEMENT KEY;
 SET NEW ELEMENT LINK TO HEAD POINTER$;
 SET HEAD POINTER TO NEW ELEMENT$;
 DISPLAY FUNCTION COMPLETE MESSAGE$;
```

APPENDIX C

Data for program COUNT

HALSTEAD'S COMPLEXITY MEASURES FOR COUNT PLI PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 50

LENGTH = N = N1 + N2 = 1845

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 233.1

VOLUME = V = N LOG2 ETA = 10412.9

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0022

MOST COMPACT VOLUME = V* = LV = 22.9

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.05

MENTAL EFFORT = E = V/L = 4733143.0

TIME (IN MINUTES) = T = E / (S * 1080) 4382.5

HALSTEAD'S COMPLEXITY MEASURES FOR COUNT PLI PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 50

LENGTH = N = N1 + N2 = 1845

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 233.1

VOLUME = V = N LOG2 ETA = 10412.9

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1)(ETA-2/N2) = 0.0022

MOST COMPACT VOLUME = V* = LV = 22.9

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.05

MENTAL EFFORT = E = V/L = 4733143.0

TIME (IN MINUTES) = T = E / (S * 1080) 4382.5

HALSTEAD'S COMPLEXITY MEASURES FOR DIAGRAM COUNT

VOCABULARY = ETA = ETA-1 + ETA-2 = 97

LENGTH = N = N1 + N2 = 580

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 551.5

VOLUME = V = N LOG2 ETA = 3827.9

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1)(ETA-2/N2) = 0.0156

MOST COMPACT VOLUME = V* = LV = 59.6

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.93

MENTAL EFFORT = E = V/L = 246012.2

TIME (IN MINUTES) = T = E / (S * 60) = 227.8

TABLE 1. OPERATORS OF DIAGRAM COUNT

		OPERATOR	COUNT
	1	BRACE	50
	2	.BEGIN	9
	3	SAVE	1
	4	FOR	5
	5	SET	5
	6	OF	16
	7	TO	12
	8	FOR EVERY/EACH/ALL	2
	9	()	35
	10	,	35
	11	.END	10
	12	CREATE	2
	13	FROM	4
	14	READ	2
	15	=	33
	16	OR	17
	17	NOT	15
	18	GET	3
	19	IN	3
	20	CALL	20
	21	WITH	20
	22	.SKIP	3
	23	REMOVE	1
	24	SEARCH	4
	25	PLUS	1
	26	INCREMENT	2
	27	ADD	2
	28	ASK	1
	29	TO INDICATE	1
	30	ABORT	1
	31	PRINT	3
ETA-1 =	32	CALCULATE	1
			319 = N1

TABLE 2. OPERANDS OF DIAGRAM COUNT

	OPERAND	COUNT
1	DIAGRAM COUNT	1
2	PROCEDURE SETUP	1
3	DIAGRAM TITLE	2
4	OUTPUT TABLES	1
5	HEAD	2
6	LINKED LIST	10
7	OPERANDS	3
8	OPERATORS	5
9	'BRACE'	2
10	LINE	1
11	0	35
12	X	1
13	PROCEDURE PRINT	1
14	PREPOSITIONS/CONNECTIVES	2
15	INPUT FILE	2
16	INFINITIVE PHRASES	2
17	INPUT LINE	6
18	FIRST CHAR	5
19	BLANK	2
20	1	35
21	BRACE INDICATOR	2
22	TRUE	15
23	FIRST WORD	12
24	DOT OPERATOR	3
25	PROCEDURE LOGOPR	12
26	'+'	4
27	'OR'	1
28	'FOR'	2
29	'FOR EVERY/EACH/ALL'	2
30	PROCEDURE LOGOPD	8
31	OBJECT	1
32	PROCEDURE RANGE	3
33	RANGE SYMBOL	2
34	NEXT LINE	2
35	WORD	1
36	W	1
37	PROCEDURE BRANCH	1
38	PROCEDURE NEXTWD	5
39	NEXT WORD	7
40	LAST WORD	6
41	LEFTMOST WORD	1
42	MATCH	12
43	MATCHED WORD	7
44	'TO'	2
45	PROCEDURE PROBLEM	2

	46	PROCEDURE ENDING	1
	47	PHRASE	3
	48	PARAMETER	4
	49	OPERATOR COUNT	1
	50	COUNT	2
	51	OPERAND COUNT	1
	52	'()'	1
	53	','	1
	54	FIRST RANGE VALUE	1
	55	SECOND RANGE VALUE	1
	56	BRANCH TEST VALUE	1
	57	BRANCH TEST OPERATOR	1
	58	UNKNOWN PREPOSITION	1
	59	UNKNOWN INFINITIVE PHRASE	1
	60	UNPROCESSIBLE LINE	1
	61	PREPOSITION/CONNECTIVE	1
	62	INFINITIVE PHRASE	1
	63	PROGRAM	1
	64	TABLE	2
ETA-2 =	65	COMPLEXITY VALUES	2
			261 = N2

```
COUNT;
 .BEGIN$;
 SETUP;
 SAVE DIAGRAM TITLE FOR OUTPUT TABLES$;
 SET HEAD OF LINKED LIST OF OPERANDS TO DIAGRAM TITLE$;
 SET HEAD OF LINKED LIST OF OPERATORS TO 'BRACE'$;
 FOR EVERY LINE #0-X;
 PRINT;
 .END$;
SETUP;
 CREATE LINKED LIST OF PREPOSITIONS/CONNECTIVES FROM INPUT FILE$;
 CREATE LINKED LIST OF INFINITIVE PHRASES FROM INPUT FILE$;
FOR EVERY LINE;
 READ INPUT LINE$;
 FIRST CHAR = BLANK #0-1;
 + ¬ FIRST CHAR = BLANK #0-1;
FIRST CHAR = BLANK;
 BRACE INDICATOR #0-1;
 + ¬ BRACE INDICATOR #0-1;
 GET FIRST WORD IN INPUT LINE$;
 FIRST WORD = DOT OPERATOR #0-1;
 + ¬ FIRST WORD = DOT OPERATOR #0-1;
BRACE INDICATOR #0-1;
 CALL PROCEDURE LOGOPR WITH 'BRACE'$;
¬ BRACE INDICATOR;
 .SKIP$;
FIRST WORD = DOT OPERATOR;
 CALL PROCEDURE LOGOPR WITH DOT OPERATOR$;
¬ FIRST WORD = DOT OPERATOR;
 FIRST WORD = '+' #0-1;
 + ¬ FIRST WORD = '+' #0-1;
FIRST WORD = '+';
 CALL PROCEDURE LOGOPR WITH 'OR'$;
¬ FIRST WORD = '+';
 FIRST WORD = 'FOR' #0-1;
 + ¬ FIRST WORD = 'FOR' #0-1;
FIRST WORD = 'FOR';
 CALL PROCEDURE LOGOPR WITH 'FOR EVERY/EACH/ALL'$;
 CALL PROCEDURE LOGOPD WITH OBJECT OF 'FOR EVERY/EACH/ALL'$;
 RANGE;
¬ FIRST WORD = 'FOR';
 RANGE SYMBOL #0-1;
 + ¬ RANGE SYMBOL #0-1;
RANGE SYMBOL;
 READ NEXT LINE$;
 GET FIRST CHAR OF NEXT LINE$;
 FIRST CHAR = '+' #0-1;
 + ¬ FIRST CHAR = '+' #0-1;
¬ RANGE SYMBOL;
 FOR EVERY WORD #0-W;
FIRST CHAR = '+';
 BRANCH;
¬ FIRST CHAR = '+';
 CALL PROCEDURE LOGOPD WITH FIRST WORD$;
 NEXTWD;
```

```
 RANGE;
FOR EVERY WORD;
 CALL PROCEDURE LOGOPR WITH FIRST WORD$;
 NEXTWD;
 NEXT WORD = LAST WORD #0-1;
 + ¬ NEXT WORD = LAST WORD #0-1;
NEXTWD;
 .BEGIN$;
 GET NEXT WORD IN INPUT LINE$;
 REMOVE LEFTMOST WORD FROM INPUT LINE$;
 .END$;
NEXT WORD = LAST WORD;
 CALL PROCEDURE LOGOPD WITH LAST WORD$;
¬ NEXT WORD = LAST WORD;
 SEARCH LINKED LIST OF PREPOSITIONS/CONNECTIVES FOR MATCH TO NEXT WORD$;
 MATCH #0-1;
 + ¬ MATCH #0-1;
MATCH;
 MATCHED WORD = 'TO' #0-1;
 + ¬ MATCHED WORD = 'TO' #0-1;
¬ MATCH;
 PROBLEM;
MATCHED WORD = 'TO';
 FNDINF;
¬ MATCHED WORD = 'TO';
 CALL PROCEDURE LOGOPR WITH MATCHED WORD$;
 MATCHED WORD = FIRST WORD #0-1;
 + ¬ MATCHED WORD = FIRST WORD #0-1;
MATCHED WORD = FIRST WORD;
 .SKIP$;
¬ MATCHED WORD = FIRST WORD;
 CALL PROCEDURE LOGOPD WITH INPUT LINE FROM FIRST WORD TO MATCHED WORD$;
 NEXTWD;
 NEXT WORD = LAST WORD #0-1;
 + ¬ NEXT WORD = LAST WORD #0-1;
NEXT WORD = LAST WORD #0-1;
 CALL PROCEDURE LOGOPD WITH LAST WORD$;
¬NEXT WORD = LAST WORD #0-1;
 .SKIP$;
FNDINF;
 .BEGIN$;
 SET PHRASE TO MATCHED WORD PLUS NEXT WORD IN INPUT LINE$;
 SEARCH LINKED LIST OF INFINITIVE PHRASES FOR MATCH TO PHRASE$;
 MATCH #0-1;
 + ¬ MATCH #0-1;
 .END$;
FNDINF.MATCH;
 CALL PROCEDURE LOGOPR WITH PHRASE$;
 NEXTWD;
 NEXTWD;
FNDINF.¬MATCH;
 PROBLEM;
 .END$;
LOGOPR;
 .BEGIN$;
```

```
 SEARCH LINKED LIST OF OPERATORS FOR MATCH TO PARAMETER$;
 MATCH #0-1;
 + ¬ MATCH #0-1;
 .END$;
LOGOPR.MATCH;
 INCREMENT OPERATOR COUNT$;
LOGOPR.¬ MATCH;
 ADD PARAMETER TO LINKED LIST OF OPERATORS$;
 SET COUNT TO 1$;
LOGOPD;
 .BEGIN$;
 SEARCH LINKED LIST OF OPERANDS FOR MATCH TO PARAMETER$;
 MATCH #0-1;
 + ¬ MATCH #0-1;
 .END$;
LOGOPD.MATCH;
 INCREMENT OPERAND COUNT$;
LOGOPD.¬ MATCH;
 ADD PARAMETER TO LINKED LIST OF OPERANDS$;
 SET COUNT TO 1$;
RANGE;
 .BEGIN$;
 CALL PROCEDURE LOGOPR WITH '()'$;
 CALL PROCEDURE LOGOPR WITH ','$;
 CALL PROCEDURE LOGOPD WITH FIRST RANGE VALUE$;
 CALL PROCEDURE LOGOPD WITH SECOND RANGE VALUE$;
 .END$;
BRANCH;
 .BEGIN$;
 CALL PROCEDURE LOGOPD WITH BRANCH TEST VALUE$;
 CALL PROCEDURE LOGOPR WITH BRANCH TEST OPERATOR$;
 RANGE;
 .END$;
PROBLEM;
 .BEGIN$;
 ASK TERMINAL OPERATOR TO INDICATE PROBLEM$;
 UNKNOWN PREPOSITION #0-1;
 + UNKNOWN INFINITIVE PHRASE #0-1;
 + UNPROCESSIBLE LINE #0-1;
 .END$;
PROBLEM.UNKNOWN PREPOSITION/CONNECTIVE;
 CALL PROCEDURE LOGOPR WITH PREPOSITION/CONNECTIVE$;
PROBLEM.UNKNOWN INFINITIVE PHRASE;
 CALL PROCEDURE LOGOPR WITH INFINITIVE PHRASE$;
PROBLEM.UNPROCESSIBLE LINE;
 ABORT PROGRAM$;
PRINT;
 .BEGIN$;
 PRINT TABLE OF OPERATORS$;
 PRINT TABLE OF OPERATORS$;
 CALCULATE COMPLEXITY VALUES$;
 PRINT COMPLEXITY VALUES$;
 .END$;
```

APPENDIX D

Data for program SORT1

HALSTEAD'S COMPLEXITY MEASURES FOR DIAGRAM SORT1

VOCABULARY = ETA = ETA-1 + ETA-2 = 32

LENGTH = N = N1 + N2 = 76

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 128.8

VOLUME = V = N LOG2 ETA = 380.0

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0402

MOST COMPACT VOLUME = V* = LV = 15.3

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.62

MENTAL EFFORT = E = V/L = 9443.3

TIME (IN MINUTES) = T = E / (S * 60) = 8.7

HALSTEAD'S COMPLEXITY MEASURES FOR SORT1 PLDS PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 22

LENGTH = N = N1 + N2 = 87

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 76.2

VOLUME = V = N LOG2 ETA = 388.0

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0427

MOST COMPACT VOLUME = V* = LV = 16.6

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.71

MENTAL EFFORT = E = V/L = 9094.5

TIME (IN MINUTES) = T = E / (S * 1080) 8.4

HALSTEAD'S COMPLEXITY MEASURES FOR SORT1 ASSEMBLER PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 50

LENGTH = N = N1 + N2 = 159

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 232.7

VOLUME = V = N LOG2 ETA = 897.4

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0218

MOST COMPACT VOLUME = V* = LV = 19.5

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.43

MENTAL EFFORT = E = V/L = 41201.7

TIME (IN MINUTES) = T = E / (S * 1080) 38.1

TABLE 1. OPERATORS OF DIAGRAM SORT1

		OPERATOR	COUNT
	1	BRACE	7
	2	.BEGIN	1
	3	SETON	2
	4	SET	2
	5	TO	2
	6	MINUS	1
	7	FOR EVERY/EACH/ALL	2
	8	()	6
	9	,	6
	10	.END	1
	11	=	2
	12	OR	2
	13	SETOFF	1
	14	DECREMENT	1
	15	.SKIP	2
	16	>	1
	17	<	1
	18	SWAP	1
ETA-1 =	19	AND	1
			42 = N1

TABLE 2. OPERANDS OF DIAGRAM SORT1

		OPERAND	COUNT
	1	DIAGRAM SORT1	1
	2	NATURAL ORDER SWITCH	5
	3	LOOP VARIABLE	3
	4	LIST LENGTH	2
	5	MAXIMUM PASSES	2
	6	1	5
	7	SORTING PASS	1
	8	0	6
	9	'1'B	1
	10	'0'B	1
	11	LIST ITEM	1
	12	PRESENT LIST ITEM	3
ETA-2 =	13	NEXT LIST ITEM	3
			34 = N2

FILE: SORT1 WORKS A

```
SORT1;
 .BEGIN$;
 SETON NATURAL ORDER SWITCH$;
 SET LOOP VARIABLE TO LIST LENGTH$;
 SET MAXIMUM PASSES TO LIST LENGTH MINUS 1$;
 FOR EVERY SORTING PASS #0-MAXIMUM PASSES;
 .END$;
FOR EVERY SORTING PASS;
 NATURAL ORDER SWITCH = '1'B #0-1;
 + NATURAL ORDER SWITCH = '0'B #0-1;
NATURAL ORDER SWITCH = '1'B;
 SETOFF NATURAL ORDER SWITCH$;
 DECREMENT LOOP VARIABLE$;
 FOR EVERY LIST ITEM #0-LOOP VARIABLE;
SWITCH = '0'B;
 .SKIP$;
FOR EVERY LIST ITEM;
 PRESENT LIST ITEM > NEXT LIST ITEM #0-1;
 + PRESENT LIST ITEM < NEXT LIST ITEM #0-1;
PRESENT LIST ITEM > NEXT LIST ITEM;
 SETON NATURAL ORDER SWITCH$;
 SWAP PRESENT LIST ITEM AND NEXT LIST ITEM$;
PRESENT LIST ITEM < NEXT LIST ITEM;
 .SKIP$;
```

APPENDIX E

Data for program SORT2

HALSTEAD'S COMPLEXITY MEASURES FOR DIAGRAM SORT2

VOCABULARY = ETA = ETA-1 + ETA-2 = 36

LENGTH = N = N1 + N2 = 89

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 150.8

VOLUME = V = N LOG2 ETA = 460.1

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0357

MOST COMPACT VOLUME = V* = LV = 16.4

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.59

MENTAL EFFORT = E = V/L = 12885.0

TIME (IN MINUTES) = T = E / (S * 60) = 11.9

HALSTEAD'S COMPLEXITY MEASURES FOR SORT2 PLI PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 37

LENGTH = N = N1 + N2 = 222

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 155.8

VOLUME = V = N LOG2 ETA = 1156.5

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0231

MOST COMPACT VOLUME = V* = LV = 26.7

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.62

MENTAL EFFORT = E = V/L = 50043.2

TIME (IN MINUTES) = T = E / (S * 1080) 46.3

TABLE 1. OPERATORS OF DIAGRAM SORT2

		OPERATOR	COUNT
	1	BRACE	7
	2	SETOFF	2
	3	SET	4
	4	TO	4
	5	CALL	2
	6	WITH	2
	7	AND	3
	8	=	2
	9	()	5
	10	,	5
	11	OR	2
	12	NOT	1
	13	.END	2
	14	.BEGIN	1
	15	FOR EVERY/EACH/ALL	1
	16	<	1
	17	>	1
	18	SETON	1
	19	SWAP	1
	20	DECREMENT	1
ETA-1 =	21	.SKIP	1
			49 = N1

TABLE 2. OPERANDS OF DIAGRAM SORT2

		OPERAND	COUNT
	1	DIAGRAM SORT2	1
	2	DO AGAIN FLAG	3
	3	PRESENT POINTER	6
	4	LIST HEAD	2
	5	TAIL POINTER	5
	6	LIST END	1
	7	PROCEDURE BUBBLE	2
	8	DO AGAIN FLAG ON	2
	9	TRUE	2
	10	0	4
	11	1	4
	12	LIST ITEM	1
	13	PRESENT ITEM	3
	14	NEXT ITEM	3
ETA-2 =	15	NEXT POINTER	1
			40 = N2

```
FILE: SORT2    WORKS    A

SORT2;
 SETOFF DO AGAIN FLAG$;
 SET PRESENT POINTER TO LIST HEAD$;
 SET TAIL POINTER TO LIST END$;
 BUBBLE;
 DO AGAIN FLAG ON #0-1;
 + ¬ DO AGAIN FLAG ON #0-1;
 .END$;
BUBBLE;
 .BEGIN$;
 FOR EVERY LIST ITEM #PRESENT POINTER-TAIL POINTER;
 .END$;
BUBBLE.FOR EVERY LIST ITEM;
 PRESENT ITEM < NEXT ITEM #0-1;
 + PRESENT ITEM > NEXT ITEM #0-1;
BUBBLE.FOR EVERY LIST ITEM.PRESENT ITEM < NEXT ITEM;
 SETON DO AGAIN FLAG$;
 SWAP PRESENT ITEM AND NEXT ITEMS$;
BUBBLE.FOR EVERY LIST ITEM.PRESENT ITEM > NEXT ITEM;
 SET PRESENT POINTER TO NEXT POINTER$;
DO AGAIN FLAG ON;
 SETOFF DO AGAIN FLAG$;
 DECREMENT TAIL POINTER$;
 SET PRESENT POINTER TO LIST HEAD$;
 BUBBLE;
¬DO AGAIN FLAG ON;
 .SKIP$;
```

APPENDIX F

Data for program BKB2PFGP

HALSTEAD'S COMPLEXITY MEASURES FOR DIAGRAM BKB2PFGP

VOCABULARY = ETA = ETA-1 + ETA-2 = 62

LENGTH = N = N1 + N2 = 352

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 314.0

VOLUME = V = N LOG2 ETA = 2095.9

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0290

MOST COMPACT VOLUME = V* = LV = 60.8

LANGUAGE LEVEL = GAMMA = (L**2) * V = 1.76

MENTAL EFFORT = E = V/L = 72246.7

TIME (IN MINUTES) = T = E / (S * 60) = 66.9

HALSTEAD'S COMPLEXITY MEASURES FOR BKB2PFGP PLDS PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 82

LENGTH = N = N1 + N2 = 542

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 451.1

VOLUME = V = N LOG2 ETA = 3445.8

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1)(ETA-2/N2) = 0.0228

MOST COMPACT VOLUME = V* = LV = 78.5

LANGUAGE LEVEL = GAMMA = (L**2) * V = 1.79

MENTAL EFFORT = E = V/L = 151264.0

TIME (IN MINUTES) = T = E / (S * 1080) 140.1

HALSTEAD'S COMPLEXITY MEASURES FOR BKB2PFGP ASSEMBLER PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 123

LENGTH = N = N1 + N2 = 957

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 740.0

VOLUME = V = N LOG2 ETA = 6644.0

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0088

MOST COMPACT VOLUME = V* = LV = 58.2

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.51

MENTAL EFFORT = E = V/L = 758445.9

TIME (IN MINUTES) = T = E / (S * 1080) 702.3

TABLE 1. OPERATORS OF DIAGRAM BKB2PFGP

		OPERATOR	COUNT
	1	BRACE	32
	2	.BEGIN	1
	3	=	28
	4	()	29
	5	,	29
	6	OR	14
	7	NOT	14
	8	FOR EVERY/EACH/ALL	1
	9	.END	1
	10	SET	3
	11	TO	3
	12	SETON	7
	13	SETOFF	9
	14	.SKIP	11
	15	CALCULATE	3
	16	POST	4
	17	IN	4
	18	USING	2
ETA-1 =	19	AND	1
			196 = N1

TABLE 2. OPERANDS OF DIAGRAM BKB2PFGP

		OPERAND	COUNT
	1	DIAGRAM BKB2PFGP	1
	2	FUNCTION REQUEST	3
	3	'GET CONTD'	3
	4	0	29
	5	1	28
	6	REQUEST	1
	7	R	1
	8	'NOT GET CONTD'	1
	9	RCB POINTER	2
	10	WORKAREA VALUE	1
	11	REQUEST BLOCK VALUE	1
	12	VALID REQUEST	4
	13	TRUE	26
	14	VALID REQUEST FLAG	2
	15	CAN READ AHEAD FLAG	4
	16	READAHEAD DONE FLAG	3
	17	NUMBER RECORDS FLAG	2
	18	OPEN TYPE FLAG	2
	19	RETURN FLAG	2
	20	VALID OPEN	2
	21	GET CONTD	4
	22	VALID RECORD NUMBER	2
	23	VERIFIABLE REQUEST	2
	24	REQUEST BYTES	1
	25	OVERLARGE GET REQUEST	4
	26	READAHEAD DONE	2
	27	PROCEDURE BKB2IH	1
	28	REQUEST VALUES	1
	29	RCB	2
	30	PROCEDURE BKB2PSRR	1
	31	RETURN CODE	1
	32	FILE RESPONSE	2
	33	READAHEAD	2
	34	CLOSE IN PROCESS	2
	35	NUMBER RECORDS	2
	36	SUBDIRECTORY	1
	37	REQUEST VALUE	1
	38	RETURN DONE	2
	39	PROCEDURE BKB2PFRG	1
	40	ERROR RETURN CODE	1
	41	DELAYED PROCESSING FLAG	1
	42	WORKAREA	1
ETA-2 =	43	PROCEDURE BKB2PFCF	1
			156 = N2

```
FILE: BKB2PFGP WORKS     A1

BKB2PFGP;
 .BEGIN$;
 FUNCTION REQUEST = 'GET CONTD' #0-1;
 + ¬ FUNCTION REQUEST = 'GET CONTD' #0-1;
 FOR EACH REQUEST #0-R;
 .END$;
FUNCTION REQUEST = 'GET CONTD';
 SET FUNCTION REQUEST TO 'NOT GET CONTD'$;
 SET RCB POINTER TO WORKAREA VALUE$;
¬FUNCTION REQUEST = 'GET CONTD';
 SET RCB POINTER TO REQUEST BLOCK VALUE$;
 VALID REQUEST #0-1;
 + ¬ VALID REQUEST #0-1;
VALID REQUEST;
 SETON VALID REQUEST FLAG$;
 SETOFF CAN READ AHEAD FLAG$;
 SETOFF READAHEAD DONE FLAG$;
 SETOFF NUMBER RECORDS FLAG$;
 SETOFF OPEN TYPE FLAG$;
 SETOFF RETURN FLAG$;
 VALID OPEN #0-1;
 + ¬ VALID OPEN #0-1;
 GET CONTD #0-1;
 + ¬ GET CONTD #0-1;
¬VALID REQUEST;
 SETOFF VALID REQUEST FLAG$;
VALID OPEN;
 .SKIP$;
¬VALID OPEN;
 SETON OPEN TYPE FLAG$;
GET CONTD;
 .SKIP$;
¬GET CONTD;
 VALID RECORD NUMBER #0-1;
 + ¬ VALID RECORD NUMBER #0-1;
VALID RECORD NUMBER;
 VERIFIABLE REQUEST #0-1;
 + ¬ VERIFIABLE REQUEST #0-1;
¬VALID RECORD NUMBER;
 .SKIP$;
VERIFIABLE REQUEST;
 CALCULATE REQUEST BYTES$;
¬VERIFIABLE REQUEST;
 .SKIP$;
GET REQUEST;
 OVERLARGE GET REQUEST #0-1;
 + ¬ OVERLARGE GET REQUEST #0-1;
¬GET REQUEST;
 .SKIP$;
OVERLARGE GET REQUEST;
 SETON NUMBER RECORDS FLAG$;
¬OVERLARGE GET REQUEST;
 .SKIP$;
FOR EACH REQUEST;
 READAHEAD DONE #0-1;
```

```
 + ¬ READAHEAD DONE #0-1;
 VALID REQUEST #0-1;
 + ¬ VALID REQUEST #0-1;
READAHEAD DONE;
 SETOFF CAN READ AHEAD FLAG$;
 SETOFF READAHEAD DONE FLAG$;
 BKB21H$;
¬READAHEAD DONE;
 .SKIP$;
VALID REQUEST;
 GET CONTD #0-1;
 + ¬ GET CONTD #0-1;
 POST REQUEST VALUES IN RCB$;
 BKB2PSRR;
 SETON RETURN FLAG$;
 POST RETURN CODE IN FILE RESPONSE$;
 READAHEAD #0-1;
 + ¬ READAHEAD #0-1;
 CLOSE IN PROCESS #0-1;
 + ¬ CLOSE IN PROCESS #0-1;
GET CONTD;
 CALCULATE NUMBER RECORDS USING RCB$;
¬GET CONTD;
 CALCULATE NUMBER RECORDS USING SUBDIRECTORY AND REQUEST VALUE$;
¬VALID REQUEST;
 RETURN DONE #0-1;
 + ¬ RETURN DONE #0-1;
 SETOFF CAN READ AHEAD FLAG$;
 OVERLARGE GET REQUEST #0-1;
 + ¬ OVERLARGE GET REQUEST #0-1;
 BKB2PFRG$;
RETURN DONE;
 .SKIP$;
¬RETURN DONE;
 POST ERROR RETURN CODE IN FILE RESPONSE$;
OVERLARGE GET REQUEST;
 SETON CAN READ AHEAD FLAG$;
 SETON DELAYED PROCESSING FLAG$;
 POST 'GET CONTD' IN WORKAREA$;
¬OVERLARGE GET REQUEST;
 .SKIP$;
READAHEAD;
 SETON READAHEAD DONE FLAG$;
¬READAHEAD;
 .SKIP$;
CLOSE IN PROCESS;
 BKB2PFCF;
¬CLOSE IN PROCESS;
 .SKIP$;
```

APPENDIX G

Data for program BKB2PIRW

HALSTEAD'S COMPLEXITY MEASURES FOR DIAGRAM BKB2PIRW

VOCABULARY = ETA = ETA-1 + ETA-2 = 51

LENGTH = N = N1 + N2 = 182

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 238.6

VOLUME = V = N LOG2 ETA = 1032.4

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0304

MOST COMPACT VOLUME = V* = LV = 31.4

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.96

MENTAL EFFORT = E = V/L = 33926.4

TIME (IN MINUTES) = T = E / (S * 60) = 31.4

HALSTEAD'S COMPLEXITY MEASURES FOR BKB2PIRW PLDS PROGRAM

VOCABULARY = ETA ETA-1 + ETA-2 = 59

LENGTH = N = N1 + N2 = 285

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 292.6

VOLUME = V = N LOG2 ETA = 1676.6

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1) (ETA-2/N2) = 0.0304

MOST COMPACT VOLUME = V* = LV = 51.0

LANGUAGE LEVEL = GAMMA = (L**2) * V = 1.55

MENTAL EFFORT = E = V/L = 55149.8

TIME (IN MINUTES) = T = E / (S * 1080) 51.1

HALSTEAD'S COMPLEXITY MEASURES FOR BKB2PIRW ASSEMBLER PROGRAM

VOCABULARY = ETA = ETA-1 + ETA-2 = 84

LENGTH = N = N1 + N2 = 398

EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 = 454.6

VOLUME = V = N LOG2 ETA = 2544.1

EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1)(ETA-2/N2) = 0.0122

MOST COMPACT VOLUME = V* = LV = 31.1

LANGUAGE LEVEL = GAMMA = (L**2) * V = 0.38

MENTAL EFFORT = E = V/L = 208195.0

TIME (IN MINUTES) = T = E / (S * 1080) 192.8

TABLE 1. OPERATORS OF DIAGRAM BKB2PIRW

		OPERATOR	COUNT
	1	BRACE	14
	2	.BEGIN	1
	3	SETOFF	5
	4	=	12
	5	()	13
	6	,	13
	7	OR	6
	8	NOT	6
	9	SET	1
	10	TO	1
	11	FOR EVERY/EACH/ALL	1
	12	POST	5
	13	IN	5
	14	GET	1
	15	FROM	1
	16	.END	1
	17	SET ON	5
	18	.SKIP	4
	19	DECREMENT	1
	20	TO PROCESS	2
	21	CALCULATE	2
	22	SETUP	1
ETA-1 =	23	FOR	1
			102 = N1

TABLE 2. OPERANDS OF DIAGRAM BKB2PIRW

		OPERAND	COUNT
	1	NEW JOB FLAG	2
	2	FIRST UNIT FLAG	3
	3	ABORT REQUESTED FLAG	2
	4	JOB END FLAG	3
	5	ABORT REQUEST	2
	6	TRUE	12
	7	0	13
	8	1	12
	9	NEXT TTHR	3
	10	STARTING TTHR	1
	11	SECTOR	1
	12	S	1
	13	SECTOR ADDRESS	1
	14	RETURN REGISTER	1
	15	RETURN ADDRESS	1
	16	WORKAREA	1
	17	NEW JOB	2
	18	WRITE REQUEST	2
	19	ADAPTER ADDRESS	1
	20	ACB	4
	21	FIRST UNIT	2
	22	SECTORS	2
	23	JOB END	2
	24	ABORT	2
	25	NEXT ADDRESS	1
	26	OPERATION	1
	27	'WRITE' REQUEST CODE	1
ETA-2 =	28	'READ' REQUEST CODE	1
			80 = N2

```
FILE: BKB2PIRW WORKS     A


BKB2PIRW;
 .BEGIN$;
 SETOFF NEW JOB FLAG$;
 SETOFF FIRST UNIT FLAG$;
 SETOFF ABORT REQUESTED FLAG$;
 SETOFF END OF JOB FLAG$;
 ABORT REQUEST #0-1;
 + ¬ ABORT REQUEST #0-1;
 SET NEXT TTHR TO STARTING TTHR$;
 FOR EVERY SECTOR #0-S;
 POST SECTOR ADDRESS IN RETURN REGISTER$;
 GET RETURN ADDRESS FROM WORKAREA$;
 BKB2PSXX$;
 .END$;
ABORT REQUEST;
 SETON ABORT REQUEST FLAG$;
 BKB2IH$;
¬ABORT REQUEST;
 .SKIP$;
FOR EVERY SECTOR;
 NEW JOB #0-1;
 + ¬ NEW JOB #0-1;
 WRITE REQUEST #0-1;
 + ¬ WRITE REQUEST #0-1;
 POST ADAPTER ADDRESS IN ACB$;
 FIRST UNIT #0-1;
 + ¬ FIRST UNIT #0-1;
 POST NEXT TTHR IN ACB$;
 BKB2PIC1$;
 DECREMENT SECTORS TO PROCESS$;
 END OF JOB #0-1;
 + ¬ END OF JOB #0-1;
 ABORT #0-1;
 + ¬ ABORT #0-1;
NEW JOB;
 SETON NEW JOB FLAG$;
 CALCULATE SECTORS TO PROCESS$;
 SETUP NEXT ADDRESS FOR OPERATION$;
 SETON FIRST UNIT FLAG$;
¬NEW JOB;
 .SKIP$;
WRITE REQUEST;
 POST 'WRITE' REQUEST CODE IN ACB$;
¬WRITE REQUEST;
 POST 'READ' REQUEST CODE IN ACB$;
FIRST UNIT;
 SETOFF FIRST UNIT FLAG$;
¬FIRST UNIT;
 CALCULATE NEXT TTHR$;
END OF JOB;
 SETON END OF JOB FLAG$;
¬END OF JOB;
 .SKIP$;
ABORT;
 SETON END OF JOB FLAG$;
```

```
FILE: BKB2FIRW WORKS     A

¬ABORT;
 .SKIP$;
```

APPENDIX G

Source Code for program COUNT

SOURCE LISTING

NUMBER

```
COUNT: PROCEDURE OPTIONS(MAIN);                                          MSP00010
/**********************************************************************/MSP00020
/* THIS PROGRAM COUNTS THE NUMBER OF UNIQUE OPERATORS AND UNIQUE      */MSP00030
/* OPERANDS AND THE TOTAL NUMBER OF OPERATORS AND OPERANDS IN A       */MSP00040
/* PROGRAM DESIGN THAT IS A WARNIER-ORR DIAGRAM PREPARED AS INPUT     */MSP00050
/* FOR ORR'S STRUCTURES PROGRAM.  OUTPUT CONSISTS OF TWO TABLES, ONE  */MSP00060
/* LISTING OPERATORS AND THE OTHER OPERANDS. IN ADDITION, HALSTEAD'S  */MSP00070
/* PROGRAM COMPLEXITY MEASURES FOR VOCABULARY, LENGTH, ESTIMATED      */MSP00080
/* LENGTH, VOLUME, ESTIMATED LEVEL OF ABSTRACTION, MOST COMPACT       */MSP00090
/* VOLUME, LANGUAGE LEVEL, MENTAL EFFORT, AND TIME AS ADAPTED TO      */MSP00100
/* WARNIER-ORR DIAGRAMS ARE COMPUTED USING THE OPERATOR AND OPERAND   */MSP00110
/* COUNTS AND ARE LISTED BELOW THE OUTPUT TABLES.                     */MSP00120
/**********************************************************************/MSP00130
/**********************************************************************/MSP00140
/*                                                                    */MSP00150
   DCL 1 OPERAND BASED(HEADOPD),                                         MSP00160
         2 OPDEOF FIXED BINARY,                                          MSP00170
         2 OPDCT FIXED BINARY,                                           MSP00180
         2 OPDNEXT PTR,                                                  MSP00190
         2 A FIXED BINARY,                                               MSP00200
         2 OPD CHAR(B REFER(A)),                                         MSP00210
         LAGOPD PTR;                                                     MSP00220
   DCL B FIXED BINARY INIT(30);                                          MSP00230
   DCL 1 OPERATOR BASED(HEADOPR),                                        MSP00240
         2 OPREOF FIXED BINARY,                                          MSP00250
         2 OPRCT FIXED BINARY,                                           MSP00260
         2 OPRNEXT PTR,                                                  MSP00270
         2 X FIXED BINARY,                                               MSP00280
         2 OPR CHAR(Y REFER(X)),                                         MSP00290
         LAGOPR PTR;                                                     MSP00300
   DCL Y FIXED BINARY INIT(30);                                          MSP00310
   DCL 1 PREPOSITION_CONNECTIVE BASED(HEADPC),                           MSP00320
         2 PCEOF FIXED BINARY,                                           MSP00330
         2 PCNEXT PTR,                                                   MSP00340
         2 R FIXED BINARY,                                               MSP00350
         2 PC CHAR(S REFER(R)),                                          MSP00360
         LAGPC PTR;                                                      MSP00370
   DCL S FIXED BINARY INIT(30);                                          MSP00380
   DCL 1 INFINITIVE BASED(HEADINF),                                      MSP00390
         2 INFEOF FIXED BINARY,                                          MSP00400
         2 INFNEXT PTR,                                                  MSP00410
         2 T FIXED BINARY,                                               MSP00420
         2 INF CHAR(U REFER(T)),                                         MSP00430
         LAGINF PTR;                                                     MSP00440
   DCL U FIXED BINARY INIT(30);                                          MSP00450
   DCL (TXTLINE,NXTLINE) CHAR(80) VARYING;                               MSP00460
```

```
     DCL (ENDLINE,ENDLOOP,NXTREAD,ENDSRCH,EOF)
         BIT(1) INIT('0'B);
     DCL (PRCFILE,IFFILE,INFILE) FILE RECORD
         ENV(F RECSIZE(80));
     DCL CUTFILE FILE STREAM OUTPUT PRINT;
     DCL (FIRSTWD,SRCHWD,TXTWD,MATCHWD,MATCHFND,TITLE,SAVEOPD,SRCHLINE)
         CHAR(60) VARYING;
  /*                                                                      */
  /**********************************************************************/
  /**********************************************************************/
  /*                                                                      */
  OPEN FILE(INFILE) INPUT;
  ON ENDFILE(INFILE) EOF = '1'B;
  CALL SETUP;                                   /* SET UP KEYWORDS LISTS  */
  READ FILE(INFILE) INTO(TXTLINE);
  TITLE = 'DIAGRAM ' || SUBSTR(TXTLINE,1,INDEX(TXTLINE,';') -1);
  ALLOCATE OPERAND SET(HEADOPD);     /* SET UP HEADS OF OPERAND AND      */
  A = LENGTH(TITLE);                 /* OPERATOR LISTS                   */
  OPD = TITLE;
  OPDEOF = 1;
  OPDCT = 1;
  ALLOCATE OPERATOR SET(HEADOPR);
  X = LENGTH('BRACE');
  OPR = 'BRACE';
  OPREOF = 1;
  OPRCT = 1;
  READ FILE(INFILE) INTO(TXTLINE);
  DO WHILE(EOF = '0'B);               /* PROCESS INPUT FILE WHILE MORE TO */
  ENDLINE = '0'B;                     /* READ                             */
    IF SUBSTR(TXTLINE,1,1) ¬= ' '
      THEN
                                      /* SKIP IF STRUCTURES HEADER LINE   */
        READ FILE(INFILE) INTO(TXTLINE);
    IF EOF = '0'B                     /* BEGIN PROCESSING STRUCTURES INPUT */
      THEN DO;                        /* FILE                             */
        ENDLINE = '0'B;
        TXTLINE = SUBSTR(TXTLINE,2);
        IF INDEX(TXTLINE,'$') = 0
          THEN
            CALL LOGOPR('BRACE');  /* LOG BRACE FOR EACH LINE THAT      */
                                   /* NEEDS ONE                         */
        TXTWD = SUBSTR(TXTLINE,1,INDEX(TXTLINE,' ') - 1);
        IF INDEX(TXTWD,';') ¬= 0            /* PICK OFF FIRST WORD IN    */
          THEN DO;                          /* LINE                      */
            TXTWD = SUBSTR(TXTWD,1,LENGTH(TXTWD) - 1);
            IF INDEX(TXTWD,'$') ¬= 0
              THEN
                TXTWD = SUBSTR(TXTWD,1,LENGTH(TXTWD) - 1);
            IF SUBSTR(TXTWD,1,1) ¬= '.'   /* LOG PROCEDURE NAME          */
```

NUMBER

```
            THEN DO;                                                      MSP00960
              CALL LOGOPD('PROCEDURE ' || TXTWD);                         MSP00970
              ENDLINE = '1'B;                                             MSP00980
            END;                                                          MSP00990
          END;                                                            MSP01000
      FIRSTWD = TXTWD;                                                    MSP01010
      IF FIRSTWD = '.BEGIN' | FIRSTWD = '.END' | FIRSTWD = '.SKIP'        MSP01020
        THEN DO;              /* LOG STANDARD STRUCTURES OPERATORS */MSP01030
          CALL LOGOPR(FIRSTWD);                                           MSP01040
          ENDLINE = '1'B;                                                 MSP01050
        END;                                                              MSP01060
      IF FIRSTWD = '+'         /* LOG STRUCTURES 'OR' OPERATOR      */ MSP01070
        THEN DO;                                                          MSP01080
          CALL LOGOPR('OR');                                              MSP01090
          CALL NEXTWD;                                                    MSP01100
          IF TXTWD = '¬'        /* LOG 'NOT' OPERATOR OCCURRING      */MSP01110
                                /* AFTER AN 'OR'                     */MSP01120
            THEN DO;                                                      MSP01130
              CALL LOGOPR('NOT');                                         MSP01140
              CALL NEXTWD;                                                MSP01150
            END;                                                          MSP01160
          CALL BRANCH;          /* CALL SUBROUTINE TO PROCESS REST OF*/MSP01170
          ENDLINE = '1'B;       /* AN 'OR' BRANCH STRUCTURES LINE    */MSP01180
        END;                                                              MSP01190
      IF FIRSTWD = 'FOR'                                                  MSP01200
        THEN DO;                /* LOG 'FOR' LOOP OPERATOR           */MSP01210
          CALL LOGOPR('FOR EVERY/EACH/ALL');                              MSP01220
          DO I = 1 TO 2;        /* GET PAST 'FOR ___'                */MSP01230
            CALL NEXTWD;                                                  MSP01240
          END;                                                            MSP01250
          CALL LOGOPD(SUBSTR(TXTLINE,1,INDEX(TXTLINE,'#') - 1));          MSP01260
          CALL RANGE;           /* CALL SUBROUTINE TO PROCESS REST OF*/MSP01270
          ENDLINE = '1'B;       /* 'FOR' LOOP LINE                   */MSP01280
        END;                                                              MSP01290
      IF ENDLINE = '0'B                                                   MSP01300
                              /* I.E., IF FIRST WORD IN LINE IS NOT*/ MSP01310
        THEN DO;                /* A STANDARD STRUCTURES OPERATOR AND*/MSP01320
                                /* SO LINE HAS NOT BEEN PROCESSED YET*/MSP01330
            IF INDEX(TXTLINE,'#') ¬= 0                                    MSP01340
              THEN DO;          /* IF THERE IS NO '#' IN INPUT LINE, */MSP01350
                READ FILE(INFILE) INTO(NXTLINE);                          MSP01360
                NXTREAD = '1'B;                    /*THEN GET NEXT LINE*/MSP01370
                IF SUBSTR(NXTLINE,2,1) = '+'                              MSP01380
                  THEN          /* IF NEXT LINE IS AN 'OR' STATEMENT,*/MSP01390
                       CALL BRANCH;  /* THEN PROCESS AS A BRANCH     */MSP01400
                                     /* ELSE PROCESS AS A SUBROUTINE */MSP01410
                                     /* CALL                         */MSP01420
                  ELSE DO;                                                MSP01430
                    CALL LOGOPD(TXTWD);                                   MSP01440
```

```
 NUMBER

   1450                    CALL NEXTWD;                                                MSP01450
   1460                    CALL RANGE;                                                 MSP01460
   1470                  END;                                                          MSP01470
   1480              END;                                                              MSP01480
   1490              ELSE DO;            /* IF NOT A BRANCH OR SUBROUTINE     */MSP01490
   1500                NXTREAD = '0'B;   /* CALL, THEN LOG FIRST WORD AS AN   */MSP01500
                                         /* OPERATOR                          */MSP01510
   1520                CALL LOGOPR(TXTWD);                                             MSP01520
   1530                CALL NEXTWD;                                                    MSP01530
   1540                IF ENDLINE = '1'B        /* THEN CONTINUE PROCESSING*/  MSP01540
                         THEN                                                          MSP01550
                           CALL LOGOPD(TXTWD);                                         MSP01560
   1570                DO WHILE (ENDLINE = '0'B);                                      MSP01570
   1580                  SRCHLINE = TXTLINE;                                           MSP01580
   1590                  SRCHWD = TXTWD;                                               MSP01590
   1600                  ENDLOOP = '0'B;                                               MSP01600
   1610                  MATCHFND = '0'B;                                              MSP01610
   1620                  ENDSRCH = '0'B;                                               MSP01620
   1630                  DO WHILE (ENDSRCH = '0'B);                                    MSP01630
   1640                    ENDLOOP = '0'B;                                             MSP01640
                                      /* SEARCH REST OF LINE FOR MATCH TO  */MSP01650
                                      /* LIST OF PREPOSITIONS AND          */MSP01660
                                      /* CONNECTIVES                       */MSP01670
   1680                    LAGPC = HEADPC;                                             MSP01680
   1690                    DO WHILE(ENDLOOP = '0'B);                                   MSP01690
                                      /* SEARCH THROUGH LIST ONCE FOR EACH */MSP01700
                                      /* WORD                              */MSP01710
   1720                      IF TXTWD = LAGPC->PC                                      MSP01720
                               THEN DO;                                                MSP01730
   1740                          ENDLOOP = '1'B;                                       MSP01740
   1750                          MATCHFND = '1'B;                                      MSP01750
   1760                          MATCHWD = TXTWD;                                      MSP01760
   1770                          ENDSRCH = '1'B;                                       MSP01770
   1780                        END;                                                    MSP01780
   1790                      IF LAGPC->PCEOF = 1                                       MSP01790
                               THEN                                                    MSP01800
                                 ENDLOOP = '1'B;                                       MSP01810
   1820                        ELSE                                                    MSP01820
                                 LAGPC = LAGPC->PCNEXT;                                MSP01830
   1840                    END; /* DO WHILE ENDLOOP = '0'B)*/                          MSP01840
   1850                    IF ENDLINE = '0'B & ENDSRCH = '0'B                          MSP01850
                                     /* IF PREVIOUS CALL TO NEXTWD SET THE*/ MSP01860
                                     /* END OF LINE FLAG THEN THE SEARCH  */ MSP01870
                                     /* FOR A PREPOSITION OR CONNECTIVE IN*/ MSP01880
                                     /* THE LINE HAS FAILED               */ MSP01890
                      THEN CALL NEXTWD;                                                MSP01900
   1910               ELSE ENDSRCH = '1'B;                                             MSP01910
   1920             END;  /* DO WHILE(ENDSRCH = '0'B)*/                                MSP01920
   1930             IF MATCHFND = '1'B                                                 MSP01930
```

NUMBER

```
                    THEN DO;                /* IF MATCHED WORD IS      */  MSP01940
                      IF MATCHWD = 'TO'     /* 'TO', THEN CALL SUB-    */  MSP01950
                        THEN CALL FNDINF;/* ROUTINE TO FIND OUT IF */  MSP01960
                                            /* IT IS THE BEGINNING OF */  MSP01970
                                            /* AN INFINITIVE PHRASE   */  MSP01980
                        ELSE DO;                                          MSP01990
                          CALL LOGOPR(MATCHWD);                           MSP02000
                                            /* IF MATCHED WORD IS NOT */  MSP02010
                                            /* 'TO', THEN LOG IT AS   */  MSP02020
                                            /* AN OPERATOR            */  MSP02030
                      IF INDEX(SRCHLINE,MATCHWD) > 1                      MSP02040
                        THEN                                              MSP02050
                                            /* IF MATCHED WORD IS NOT */  MSP02060
                                            /* FIRST WORD IN THE LINE */  MSP02070
                                            /* SEGMENT TESTED, THEN   */  MSP02080
                                            /* LOG THE LINE FROM THE  */  MSP02090
                                            /* BEGINNING TO BEFORE THE*/  MSP02100
                                            /* PREPOSITION OR CONNECT-*/  MSP02110
                                            /* IVE AS AN OPERAND      */  MSP02120
            CALL LOGOPD(SUBSTR(SRCHLINE,1,INDEX(SRCHLINE,MATCHWD) -2));  MSP02130
                                            /* SET TXTLINE AND        */  MSP02140
                                            /* TXTWD TO UNPROCESSED   */  MSP02150
                                            /* REMAINDER OF LINE      */  MSP02160
                      CALL NEXTWD;                                        MSP02170
                      IF ENDLINE = '1'B                                   MSP02180
                        THEN CALL LOGOPD(TXTWD);                          MSP02190
                      END;                                                MSP02200
                      END;                                                MSP02210
                      ELSE                                                MSP02220
                        CALL PROBLEM;                                     MSP02230
                END; /*DO WHILE(ENDLINE = '0'B)*/                         MSP02240
             END; /* IF # NOT FOUND IN LINE */                            MSP02250
         END; /*IF FIRST WORD IN LINE NOT STRUCTURES STANDARD*/           MSP02260
                /*OPERATOR*/                                              MSP02270
         IF NXTREAD = '1'B                                                MSP02280
           THEN DO;                                                       MSP02290
             NXTREAD = '0'B;                                              MSP02300
             TXTLINE = NXTLINE;                                           MSP02310
           END;                                                           MSP02320
           ELSE                                                           MSP02330
             READ FILE(INFILE) INTO(TXTLINE);                             MSP02340
       END;/*IF STRUCTURES INPUT LINE AND NOT EOF */                      MSP02350
     END; /*DO WHILE(EOF = '0'B)*/                                        MSP02360
   CALL PRINT;                                                            MSP02370
   /**********************************************************************/MSP02380
   /* SUBROUTINE SETUP CREATES LINKED LISTS OF PREPOSITIONS AND          */MSP02390
   /* CONNECTIVES AND OF INFINITIVE PHRASES ALL LIKELY TO BE             */MSP02400
   /* FOUND IN STRUCTURES DESIGN CHARTS.  THESE LISTS ARE USED AS        */MSP02410
   /* CHECKS AGAINST WORDS AND PHRASES OF THE INPUT LINES.               */MSP02420
```

NUMBER

```
/*********************************************************************/MSP02430
SETUP: PROCEDURE;                                                      MSP02440
DCL (FLAG1,FLAG2) BIT(1) INIT('1'B);                                   MSP02450
ON ENDFILE(PRCFILE) FLAG1 = '0'B;                                      MSP02460
ON ENDFILE(IFFILE) FLAG2 = '0'B;                                       MSP02470
ALLOCATE PREPOSITION_CONNECTIVE;                                       MSP02480
LAGPC = HEADPC;                                                        MSP02490
OPEN FILE(PRCFILE) INPUT;                                              MSP02500
READ FILE(PRCFILE) INTO(TXTLINE);                                      MSP02510
DO WHILE(FLAG1 = '1'B);                                                MSP02520
  LAGPC->PCEOF = 0;                                                    MSP02530
  TXTWD = SUBSTR(TXTLINE,1,INDEX(TXTLINE, ' ') - 1);                   MSP02540
  LAGPC->R = LENGTH(TXTWD);                                            MSP02550
  LAGPC->PC =TXTWD;                                                    MSP02560
  READ FILE(PRCFILE) INTO(TXTLINE);                                    MSP02570
  IF FLAG1 = '1'B THEN DO;                                             MSP02580
    ALLOCATE PREPOSITION_CONNECTIVE SET(LAGPC->PCNEXT);                MSP02590
    LAGPC = LAGPC->PCNEXT;                                             MSP02600
  END;                                                                 MSP02610
END;                                                                   MSP02620
LAGPC->PCEOF = 1;                                                      MSP02630
LAGPC = HEADPC;                                                        MSP02640
ALLOCATE INFINITIVE;                                                   MSP02650
LAGINF = HEADINF;                                                      MSP02660
OPEN FILE(IFFILE) INPUT;                                               MSP02670
READ FILE(IFFILE) INTO(TXTLINE);                                       MSP02680
DO WHILE(FLAG2 = '1'B);                                                MSP02690
  LAGINF->INFEOF = 0;                                                  MSP02700
  TXTWD = SUBSTR(TXTLINE,1,INDEX(TXTLINE, '  ') - 1);                  MSP02710
  LAGINF->T = LENGTH(TXTWD);                                           MSP02720
  LAGINF->INF = TXTWD;                                                 MSP02730
  READ FILE(IFFILE) INTO(TXTLINE);                                     MSP02740
  IF FLAG2 = '1'B THEN DO;                                             MSP02750
    ALLOCATE INFINITIVE SET(LAGINF->INFNEXT);                          MSP02760
    LAGINF = LAGINF->INFNEXT;                                          MSP02770
  END;                                                                 MSP02780
END;                                                                   MSP02790
LAGINF->INFEOF = 1;                                                    MSP02800
LAGINF = HEADINF;                                                      MSP02810
END SETUP;                                                             MSP02820
/*********************************************************************/MSP02830
/*********************************************************************/MSP02840
/* PROCEDURE NEXTWD ASSIGNS TO THE VARIABLE TEXTWD THE NEXT WORD IN  */MSP02850
/* THE LINE AFTER THE PRESENT VALUE OF TEXTWD.  IF THE NEW VALUE OF  */MSP02860
/* TXTWD IS THE LAST WORD IN THE LINE, THE 'ENDLINE' OPERATOR IS     */MSP02870
/* LOGGED IN THE OPERATORS LIST.                                     */MSP02880
/*********************************************************************/MSP02890
NEXTWD: PROCEDURE;                                                     MSP02900
 TXTLINE = SUBSTR(TXTLINE,LENGTH(TXTWD) + 2);                          MSP02910
```

```
 NUMBER

  2920    TXTWD = SUBSTR(TXTLINE,1,INDEX(TXTLINE,' ') - 1);                          MSP02920
  2930    IF INDEX(TXTWD,';') ¬= 0                                                   MSP02930
            THEN DO;                                                                 MSP02940
  2950        ENDLINE = '1'B;                                                        MSP02950
  2960        IF INDEX(TXTWD,'$') ¬= 0                                               MSP02960
                THEN TXTWD = SUBSTR(TXTWD,1,LENGTH(TXTWD) - 2);                      MSP02970
  2980          ELSE TXTWD = SUBSTR(TXTWD,1,LENGTH(TXTWD) - 1);                      MSP02980
  2990      END;                                                                     MSP02990
  3000    END NEXTWD;                                                                MSP03000
        /*******************************************************************/MSP03010
        /*******************************************************************/MSP03020
        /* PROCEDURE BRANCH LOGS THE OPERATORS AND OPERANDS IN A WARNIER-ORR */MSP03030
        /* 'EITHER/OR' STATEMENT OCCURRING BEFORE THE RANGE DESCRIPTION.     */MSP03040
        /* THE LATTER ARE LOGGED BY A CALL TO PROCEDURE RANGE.               */MSP03050
        /*******************************************************************/MSP03060
  3070  BRANCH: PROCEDURE;                                                           MSP03070
  3080  DCL CHKSTRING CHAR(1);                                                       MSP03080
  3090  DCL SAVEOPD CHAR(80) VARYING;                                                MSP03090
  3100  DCL OPRINDEX FIXED BINARY INIT(0);                                           MSP03100
  3110  IF INDEX(TXTLINE,'=') = 0 & INDEX(TXTLINE,'>') = 0 &                         MSP03110
          INDEX(TXTLINE,'<') = 0                                                     MSP03120
          THEN DO;                   /* IF NO COMPARISON OPERATOR IN LINE, THEN */MSP03130
                                     /* CONSTRUCTION MUST BE, FOR EXAMPLE,      */MSP03140
                                     /* 'MATCH FOUND #0-1', WHICH IS LOGGED AS  */MSP03150
                                     /* 'MATCH  FOUND = TRUE . . .'             */MSP03160
  3170      CALL LOGOPD(SUBSTR(TXTLINE,1,INDEX(TXTLINE,'#') - 2));                   MSP03170
  3180      CALL LOGOPR('=');                                                        MSP03180
  3190      CALL LOGOPD('TRUE');                                                     MSP03190
  3200    END;                                                                       MSP03200
  3210    ELSE DO;                   /* IF COMPARISON OPERATOR FOUND IN LINE,   */MSP03210
                                     /* THEN LOG LINE UP TO OPERATOR IN OPERANDS*/MSP03220
                                     /* LIST, LOG OPERATOR OR OPERATORS IN OPERA*/MSP03230
                                     /* TORS LIST (THERE MAY BE TWO, AS IN      */MSP03240
                                     /* '>='), AND MOVE BEGINNING OF TXTLINE    */MSP03250
                                     /* VARIABLE PAST OPERATOR(S)               */MSP03260
  3270      SAVEOPD = TXTLINE;                                                       MSP03270
  3280      DO WHILE(INDEX(TXTWD,'=') ¬= 1 & INDEX(TXTWD,'>') ¬= 1 &                 MSP03280
                     INDEX(TXTWD,'<') ¬= 1);                                         MSP03290
  3300        CALL NEXTWD;                                                           MSP03300
  3310      END;                                                                     MSP03310
  3320      CALL LOGOPR(SUBSTR(TXTWD,1,1));                                          MSP03320
  3330      OPRINDEX = 1;                                                            MSP03330
  3340      IF SUBSTR(TXTWD,2,1) = '=' | SUBSTR(TXTWD,2,1) = '>' |                   MSP03340
               SUBSTR(TXTWD,2,1) = '<'                                               MSP03350
             THEN DO;                                                                MSP03360
  3370         OPRINDEX = 2;                                                         MSP03370
  3380         CALL LOGOPR(SUBSTR(TXTWD,2,1));                                       MSP03380
  3390       END;                                                                    MSP03390
  3400      SAVEOPD = SUBSTR(SAVEOPD,1,LENGTH(SAVEOPD) - LENGTH(TXTLINE) - 1); MSP03400
```

PL/I OPTIMIZING COMPILER   COUNT: PROCEDURE OPTIONS(MAIN);

NUMBER

```
       CALL LOGOPD(SAVEOPD);                                           MSP03410
       IF OPRINDEX > 0                                                 MSP03420
         THEN DO;                                                      MSP03430
           TXTLINE = SUBSTR(TXTLINE,OPRINDEX + 2);                     MSP03440
           CALL LOGOPD(SUBSTR(TXTLINE,1,INDEX(TXTLINE,'#') - 2));      MSP03450
          END;                                                         MSP03460
   END;                                                                MSP03470
 CALL RANGE;                                                           MSP03480
 RETURN;                                                               MSP03490
 END BRANCH;                                                           MSP03500
 /*********************************************************************/MSP03510
 /*********************************************************************/MSP03520
 /* PROCEDURE RANGE TRANSLATES STRUCTURES INPUT FOR 'EITHER/OR',       */MSP03530
 /* DO WHILE, AND DO UNTIL RANGES INTO WARNIER-ORR FORM AND LOGS THE   */MSP03540
 /* OPERATORS AND OPERANDS.                                            */MSP03550
 /*********************************************************************/MSP03560
 RANGE: PROCEDURE;                                                     MSP03570
 TXTLINE = SUBSTR(TXTLINE,INDEX(TXTLINE,'#') + 1);                     MSP03580
 CALL LOGOPD(SUBSTR(TXTLINE,1,INDEX(TXTLINE,'-') - 1));                MSP03590
 TXTLINE = SUBSTR(TXTLINE,INDEX(TXTLINE,'-') + 1);                     MSP03600
 CALL LOGOPD(SUBSTR(TXTLINE,1,INDEX(TXTLINE,';') - 1));                MSP03610
 CALL LOGOPR('()');                                                    MSP03620
 CALL LOGOPR(',');                                                     MSP03630
 ENDLINE = '1'B;                                                       MSP03640
 RETURN;                                                               MSP03650
 END RANGE;                                                            MSP03660
 /*********************************************************************/MSP03670
 /*********************************************************************/MSP03680
 /* PROCEDURE FNDINF SEARCHES FOR A MATCH TO THE LINKED LIST OF        */MSP03690
 /* INFINITIVE PHRASES AND LOGS IT IN THE LINKED LIST OF OPERATORS     */MSP03700
 /* IF A MATCH IS FOUND OR CALLS PROCEDURE PROBLEM IF NO MATCH IS      */MSP03710
 /* FOUND.                                                             */MSP03720
 /*********************************************************************/MSP03730
 FNDINF: PROCEDURE;                                                    MSP03740
 DCL PHRASE CHAR(40) VARYING;                                          MSP03750
 PHRASE = SUBSTR(TXTLINE, INDEX(TXTLINE,' ') + 1);                     MSP03760
 PHRASE = 'TO ' || SUBSTR(PHRASE,1,INDEX(PHRASE,' ') - 1);             MSP03770
 IF INDEX(PHRASE,';') ¬= 0                                             MSP03780
   THEN DO;                                                            MSP03790
     ENDLINE = '1'B;                                                   MSP03800
     PHRASE = SUBSTR(PHRASE,1,LENGTH(PHRASE) - 1);                     MSP03810
     IF INDEX(PHRASE,'$') ¬= 0                                         MSP03820
       THEN                                                            MSP03830
         PHRASE = SUBSTR(PHRASE,1,LENGTH(PHRASE) - 1);                 MSP03840
   END;                                                                MSP03850
 LAGINF = HEADINF;                                                     MSP03860
 DO WHILE(LAGINF->INFEOF = 0);                                         MSP03870
   IF PHRASE = LAGINF->INF                                             MSP03880
     THEN DO;                                                          MSP03890
```

PL/I OPTIMIZING COMPILER   COUNT: PROCEDURE OPTIONS(MAIN);

NUMBER

```
          CALL LOGOPR(LAGINF->INF);                                    MSP03900
          CALL NEXTWD;                                                 MSP03910
          CALL NEXTWD;                                                 MSP03920
          RETURN;                                                      MSP03930
        END;                                                           MSP03940
      IF LAGINF->INFEOF = 0                                            MSP03950
        THEN                                                           MSP03960
          LAGINF = LAGINF->INFNEXT;                                    MSP03970
    END;                                                               MSP03980
    IF PHRASE = LAGINF->INF                                            MSP03990
      THEN DO;                                                         MSP04000
        CALL LOGOPR(LAGINF->INF);                                      MSP04010
        CALL NEXTWD;                                                   MSP04020
        CALL NEXTWD;                                                   MSP04030
        RETURN;                                                        MSP04040
      END;                                                             MSP04050
      ELSE DO;                                                         MSP04060
        CALL PROBLEM;                                                  MSP04070
        RETURN;                                                        MSP04080
      END;                                                             MSP04090
    END FNDINF;                                                        MSP04100
    /*******************************************************************/MSP04110
    /*******************************************************************/MSP04120
    /* PROCEDURE PROBLEM ALLOWS THE TERMINAL OPERATOR TO INTERACTIVELY   */MSP04130
    /* PARSE THE PARTS OF WARNIER-ORR LINES THAT CANNOT OTHERWISE BE     */MSP04140
    /* PARSED BY THIS PROGRAM BECAUSE THEY CONTAIN PREPOSITIONS,         */MSP04150
    /* CONNECTIVES, OR INFINITIVE PHRASES NOT IN THE MASTER LIST,        */MSP04160
    /* BECAUSE THEY ARE SYNTACTICALLY AMBIGUOUS, OR BECAUSE THEY CONTAIN */MSP04170
    /* AN ERROR.                                                           MSP04180
    /*******************************************************************/MSP04190
    PROBLEM: PROCEDURE;                                                MSP04200
    DCL (DSPLINE, WRITEVAR,PREP) CHAR(72) VARYING;                     MSP04210
    DISPLAY(SRCHLINE);                                                 MSP04220
    DISPLAY('IF NO PREPOSITIONS, CONNECTIVES, OR INFINITIVES, ENTER "N:"');MSP04230
    DISPLAY('IF "TO" APPEARS, ENTER "I:" AND PHRASE IF INFINITIVE');   MSP04240
    DISPLAY('IF "TO" APPEARS, ENTER "P:TO" IF PREPOSITION');           MSP04250
    DISPLAY('IF OTHER PREPOSITION OR CONNECTIVE, ENTER "P:" AND WORD'); MSP04260
    DISPLAY('IF LINE IS UNPROCESSIBLE, ENTER "U:"') REPLY(DSPLINE);    MSP04270
    IF SUBSTR(DSPLINE,1,2) = 'N:'                                      MSP04280
      THEN DO;                                 /* LOG REST OF LINE IN  */MSP04290
                                               /* OPERANDS LIST        */MSP04300
        IF INDEX(SRCHLINE,'$') = 0                                     MSP04310
          THEN DO;                                                     MSP04320
            SRCHLINE = SUBSTR(SRCHLINE,1,INDEX(SRCHLINE,';') - 1);     MSP04330
            CALL LOGOPD(SRCHLINE);                                     MSP04340
            RETURN;                                                    MSP04350
          END;                                                         MSP04360
          ELSE DO;                                                     MSP04370
           SRCHLINE = SUBSTR(SRCHLINE,1,INDEX(SRCHLINE,';') - 2);      MSP04380
```

PL/I OPTIMIZING COMPILER   COUNT: PROCEDURE OPTIONS(MAIN);

NUMBER

```
        CALL LOGOPD(SRCHLINE);
        RETURN;
      END;
  END;
IF SUBSTR(DSPLINE,3) ¬= ' '
  THEN DO;
    WRITEVAR = SUBSTR(DSPLINE,3);
  END;
IF SUBSTR(DSPLINE,1,2) = 'P:'
  THEN DO;
    PREP = ' ' || WRITEVAR || ' ';
    CALL LOGOPD(SUBSTR(SRCHLINE,1,INDEX(SRCHLINE,PREP) - 1));
    CALL LOGOPR(WRITEVAR);
    TXTLINE = SUBSTR(SRCHLINE,INDEX(SRCHLINE,PREP) + 1);
    TXTLINE = SUBSTR(TXTLINE,INDEX(TXTLINE,' ') + 1);
    TXTWD = SUBSTR(TXTLINE,1,INDEX(TXTLINE,' ') - 1);
    IF INDEX(TXTWD,';') = 0
      THEN
        ENDLINE = '0'B;
      ELSE DO;
        ENDLINE = '1'B;
        TXTWD = SUBSTR(TXTWD,1,LENGTH(TXTWD) - 1);
        IF INDEX(TXTWD,'$') ¬= 0
          THEN TXTWD = SUBSTR(TXTWD,1,LENGTH(TXTWD) - 1);
        CALL LOGOPD(TXTWD);
      END;
    RETURN;
  END;
IF SUBSTR(DSPLINE,1,2) = 'I:'
  THEN DO;
    CALL LOGOPD(SUBSTR(SRCHLINE,1,INDEX(SRCHLINE,WRITEVAR) - 2));
    CALL LOGOPR(WRITEVAR);
    TXTLINE = SUBSTR(SRCHLINE,INDEX(SRCHLINE,WRITEVAR) + 3);
    TXTWD = SUBSTR(TXTLINE,1,INDEX(TXTLINE,' ') - 1);
    IF INDEX(TXTWD,';') ¬= 0
      THEN
        ENDLINE = '1'B;
       ELSE DO;
        CALL NEXTWD;
        IF ENDLINE = '1'B
         THEN
           CALL LOGOPD(TXTWD);
       END;
    RETURN;
  END;
IF SUBSTR(DSPLINE,1,2) = 'U:'
  THEN DO;
    DISPLAY('LINE UNPROCESSIBLE--PROGRAM ABORTED');
    STOP;
```

PL/I OPTIMIZING COMPILER   COUNT: PROCEDURE OPTIONS(MAIN);

NUMBER

```
  END;                                                                  MSP04880
END PROBLEM;                                                            MSP04890
/**********************************************************************/MSP04900
/* PROCEDURE LOGOPR SEARCHES FOR A MATCH TO AN OPERATOR IN THE        */MSP04910
/* LINKED LIST OF OPERATORS AND INCREMENTS THE COUNT IF A MATCH IS    */MSP04920
/* FOUND OR ADDS THE OPERATOR TO THE END OF THE LIST IF A MATCH IS    */MSP04930
/* NOT FOUND.                                                         */MSP04940
/**********************************************************************/MSP04950
/**********************************************************************/MSP04960
LOGOPR: PROCEDURE(POPR);                                                MSP04970
DCL POPR CHAR(40) VARYING;                                              MSP04980
LAGOPR = HEADOPR;                                                       MSP04990
DO WHILE(LAGOPR->OPREOF = 0);                                           MSP05000
  IF POPR = LAGOPR->OPR                                                 MSP05010
    THEN DO;                                                            MSP05020
      LAGOPR->OPRCT = LAGOPR->OPRCT + 1;                                MSP05030
      RETURN;                                                           MSP05040
      END;                                                              MSP05050
  IF LAGOPR->OPREOF = 0                                                 MSP05060
    THEN                                                                MSP05070
         LAGOPR = LAGOPR->OPRNEXT;                                      MSP05080
END;                                                                    MSP05090
IF POPR = LAGOPR->OPR                                                   MSP05100
 THEN DO;                                                               MSP05110
   LAGOPR->OPRCT = LAGOPR->OPRCT + 1;                                   MSP05120
   RETURN;                                                              MSP05130
 END;                                                                   MSP05140
   LAGOPR->OPREOF = 0;                                                  MSP05150
   ALLOCATE OPERATOR SET(LAGOPR->OPRNEXT);                              MSP05160
   LAGOPR = LAGOPR->OPRNEXT;                                            MSP05170
   LAGOPR->X = LENGTH(POPR);                                            MSP05180
   LAGOPR->OPR = POPR;                                                  MSP05190
   LAGOPR->OPRCT = 1;                                                   MSP05200
   LAGOPR->OPREOF = 1;                                                  MSP05210
   RETURN;                                                              MSP05220
END LOGOPR;                                                             MSP05230
/**********************************************************************/MSP05240
/**********************************************************************/MSP05250
/* PROCEDURE LOGOPD SEARCHES FOR A MATCH TO AN OPERAND IN THE LINKED */MSP05260
/* LIST OF OPERANDS AND INCREMENTS THE COUNT IF A MATCH IS FOUND OR  */MSP05270
/* ADDS IT TO THE END OF THE LIST IF A MATCH IS NOT FOUND.           */MSP05280
/**********************************************************************/MSP05290
LOGOPD: PROCEDURE(POPD);                                                MSP05300
DCL POPD CHAR(40) VARYING;                                              MSP05310
LAGOPD = HEADOPD;                                                       MSP05320
DO WHILE(LAGOPD->OPDEOF = 0);                                           MSP05330
  IF POPD = LAGOPD->OPD                                                 MSP05340
    THEN DO;                                                            MSP05350
      LAGOPD->OPDCT = LAGOPD->OPDCT + 1;                                MSP05360
```

PL/I OPTIMIZING COMPILER   COUNT: PROCEDURE OPTIONS(MAIN);

NUMBER

```
          RETURN;                                                         MSP05370
        END;                                                              MSP05380
      IF LAGOPD->OPDEOF = 0                                               MSP05390
        THEN                                                              MSP05400
          LAGOPD = LAGOPD->OPDNEXT;                                       MSP05410
    END;                                                                  MSP05420
    IF POPD = LAGOPD->OPD                                                 MSP05430
      THEN DO;                                                            MSP05440
        LAGOPD->OPDCT = LAGOPD->OPDCT + 1;                                MSP05450
        RETURN;                                                           MSP05460
      END;                                                                MSP05470
      ELSE DO;                                                            MSP05480
        LAGOPD->OPDEOF = 0;                                               MSP05490
        ALLOCATE OPERAND SET(LAGOPD->OPDNEXT);                            MSP05500
        LAGOPD = LAGOPD->OPDNEXT;                                         MSP05510
        LAGOPD->A = LENGTH(POPD);                                         MSP05520
        LAGOPD->OPD = POPD;                                               MSP05530
        LAGOPD->OPDCT = 1;                                                MSP05540
        LAGOPD->OPDEOF = 1;                                               MSP05550
        RETURN;                                                           MSP05560
      END;                                                                MSP05570
    END LOGOPD;                                                           MSP05580
    /*****************************************************************/MSP05590
    /*****************************************************************/MSP05600
    /* PROCEDURE PRINT PRODUCES TABLES OF OPERATOR AND OPERAND COUNTS   */MSP05610
    /* AND PRINTS OUT THE VALUES OF HALSTEAD'S COMPLEXITY MEASURES FOR  */MSP05620
    /* A WARNIER-ORR DIAGRAM.                                           */MSP05630
    /*****************************************************************/MSP05640
    PRINT: PROCEDURE;                                                     MSP05650
    DCL (TOTOPRS,TOTOPDS,OPRS,OPDS) FIXED DECIMAL(10,5);                  MSP05660
    DCL (EST_N,GAMMA,V_COM,EST_L,V,E,T,ETA,N) FIXED DECIMAL(10,5);        MSP05670
    OPEN FILE(OUTFILE) PAGESIZE(55) LINESIZE(80);                         MSP05680
    TOTOPRS = 0;                                                          MSP05690
    OPRS = 0;                                                             MSP05700
    TOTOPDS = 0;                                                          MSP05710
    OPDS = 0;                                                             MSP05720
    PUT FILE(OUTFILE) SKIP(3) EDIT('TABLE 1.  OPERATORS OF',TITLE)        MSP05730
          (COL(22),A(22),X(1),A(30));                                     MSP05740
    PUT FILE(OUTFILE) SKIP(2) EDIT                                        MSP05750
          ('____________________________________________________________')MSP05760
          (COL(7),A(72));                                                 MSP05770
    PUT FILE(OUTFILE) SKIP;                                               MSP05780
    PUT FILE(OUTFILE) SKIP EDIT('OPERATOR','COUNT')                       MSP05790
          (COL(30),A(8),X(21),A(5));                                      MSP05800
    PUT FILE(OUTFILE) SKIP EDIT                                           MSP05810
          ('____________________________________________________________')MSP05820
          (COL(7),A(72));                                                 MSP05830
    PUT FILE(OUTFILE) SKIP;                                               MSP05840
    LAGOPR = HEADOPR;                                                     MSP05850
```

PL/I OPTIMIZING COMPILER   COUNT: PROCEDURE OPTIONS(MAIN);

NUMBER

```
DO WHILE(LAGOPR->OPREOF = 0);
  TOTOPRS = TOTOPRS + LAGOPR->OPRCT;
  OPRS = OPRS + 1;
  PUT FILE(OUTFILE) SKIP EDIT(OPRS,LAGOPR->OPR,LAGOPR->OPRCT)
                (COL(17),F(3),X(4),A(25),X(10),F(3));
  LAGOPR = LAGOPR->OPRNEXT;
END;
TOTOPRS = TOTOPRS + LAGOPR->OPRCT;
OPRS = OPRS + 1;
PUT FILE(OUTFILE)
                SKIP EDIT('ETA-1 =',OPRS,LAGOPR->OPR,LAGOPR->OPRCT)
                (COL(9),A(7),X(1),F(3),X(4),A(25),X(10),F(3));
PUT FILE(OUTFILE) SKIP EDIT('___________________________________',
                '__________')(COL(7),A(35),X(15),A(10));
PUT FILE(OUTFILE) SKIP EDIT(TOTOPRS,'= N1')(COL(59),F(3),X(1),A(4));
PUT FILE(OUTFILE) PAGE;
PUT FILE(OUTFILE) SKIP(3) EDIT('TABLE 2.  OPERANDS OF',TITLE)
      (COL(22),A(21),X(1),A(30));
PUT FILE(OUTFILE) SKIP(2) EDIT
      ('________________________________________________________________________')
      (COL(7),A(72));
PUT FILE(OUTFILE) SKIP;
PUT FILE(OUTFILE) SKIP EDIT('OPERAND','COUNT')(COL(30),A,X(22),A);
PUT FILE(OUTFILE) SKIP EDIT
      ('________________________________________________________________________')
      (COL(7),A(72));
PUT FILE(OUTFILE) SKIP;
LAGOPD = HEADOPD;
DO WHILE(LAGOPD->OPDEOF = 0);
  TOTOPDS = TOTOPDS + LAGOPD->OPDCT;
  OPDS = OPDS + 1;
  PUT FILE(OUTFILE) SKIP EDIT(OPDS,LAGOPD->OPD,LAGOPD->OPDCT)
                (COL(17),F(3),X(4),A(25),X(10),F(3));
  LAGOPD = LAGOPD->OPDNEXT;
END;
TOTOPDS = TOTOPDS + LAGOPD->OPDCT;
OPDS = OPDS + 1;
PUT FILE(OUTFILE)
                SKIP EDIT('ETA-2 =',OPDS,LAGOPD->OPD,LAGOPD->OPDCT)
                (COL(9),A(7),X(1),F(3),X(4),A(25),X(10),F(3));
PUT FILE(OUTFILE) SKIP EDIT('___________________________________',
                '__________')(COL(7),A(35),X(15),A(10));
PUT FILE(OUTFILE) SKIP EDIT(TOTOPDS,'= N2')(COL(59),F(3),X(1),A(4));
PUT FILE(OUTFILE) PAGE;
PUT FILE(OUTFILE) SKIP(3) EDIT('HALSTEAD''S COMPLEXITY MEASURES FOR',
           TITLE)(COL(11),A(34),X(1),A(30));
PUT FILE(OUTFILE) SKIP(2);
ETA = OPRS + OPDS;
PUT FILE(OUTFILE) SKIP EDIT('VOCABULARY = ETA = ETA-1 + ETA-2 =',
```

PL/I OPTIMIZING COMPILER   COUNT: PROCEDURE OPTIONS(MAIN);

```
 NUMBER

                 ETA)(COL(9),A(34),F(4));                                     MSP06350
  6360   N = TOTOPRS + TOTOPDS;                                               MSP06360
  6370   PUT FILE(OUTFILE) SKIP(2) EDIT                                       MSP06370
                ('LENGTH = N = N1 + N2 =',N)                                  MSP06380
                (COL(9),A(22),F(5));                                          MSP06390
  6400   EST_N = (OPRS * LOG2(OPRS)) + (OPDS * LOG2(OPDS));                   MSP06400
  6410   PUT FILE(OUTFILE) SKIP(2) EDIT                                       MSP06410
                ('EST. N = ETA-1 LOG2 ETA-1 + ETA-2 LOG2 ETA-2 =',EST_N)      MSP06420
                (COL(9),A(46),X(1),F(5,1));                                   MSP06430
  6440   V = N * LOG2(ETA);                                                   MSP06440
  6450   PUT FILE(OUTFILE) SKIP(2) EDIT                                       MSP06450
                ('VOLUME = V = N LOG2 ETA =',V)                               MSP06460
                (COL(9),A(25),X(1),F(7,1));                                   MSP06470
  6480   EST_L = (2 / OPRS) * (OPDS / TOTOPDS);                               MSP06480
  6490   PUT FILE(OUTFILE) SKIP(2) EDIT                                       MSP06490
                ('EST. ABSTRACTION LEVEL = EST. L = (2/ETA-1)(ETA-2/N2) =',   MSP06500
                 EST_L)(COL(9),A(55),X(1),F(6,4));                            MSP06510
  6520   V_COM = EST_L * V;                                                   MSP06520
  6530   PUT FILE(OUTFILE) SKIP(2) EDIT                                       MSP06530
                ('MOST COMPACT VOLUME = V* = LV =',V_COM)                     MSP06540
                (COL(9),A(32),X(1),F(4,1));                                   MSP06550
  6560   GAMMA = (EST_L**2) * V;                                              MSP06560
  6570   PUT FILE(OUTFILE) SKIP(2) EDIT                                       MSP06570
                ('LANGUAGE LEVEL = GAMMA = (L**2) * V =',GAMMA)               MSP06580
                (COL(9),A(37),X(1),F(6,2));                                   MSP06590
  6600   E = V / EST_L;                                                       MSP06600
  6610   PUT FILE(OUTFILE) SKIP(2) EDIT                                       MSP06610
                ('MENTAL EFFORT = E = V/L =',E)                               MSP06620
                (COL(9),A(25),X(1),F(7,1));                                   MSP06630
  6640   T =  E / 1080;                                                       MSP06640
  6650   PUT FILE(OUTFILE) SKIP(2) EDIT                                       MSP06650
                ('TIME (IN MINUTES) = T = E / (S * 60) =',T)                  MSP06660
                (COL(9),A(38),X(1),F(5,1));                                   MSP06670
  6680   END PRINT;                                                           MSP06680
         /*******************************************************************/MSP06690
  6700   END COUNT;                                                           MSP06700
```

PREDICTING PROGRAM COMPLEXITY FROM WARNIER-ORR DIAGRAMS

by

BARBARA WHITE

B.A., University of Kansas, 1965
M.A., University of Missouri, 1968

ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

ABSTRACT

Halstead's complexity metrics are an objective measure of program complexity based on counts of the operators and operands in a program. They include formulas for vocabulary, length, estimated length, language level, abstraction level, mental effort, and programming time, and considerable interest has been manifested in their practical applications. In the present experiment, Halstead's metrics were adapted to Warnier-Orr diagrams of program designs, and the Halstead values for diagrams were compared to those for the programs written from them. Six WO diagrams, six high-level-language programs and three assembler-language programs were analyzed using an operator and operand counting program. A statistically significant relationship was found for diagram and high-level-language program estimated abstraction level, and values of diagram and assembler-language programs for these three metrics were also apparently related. From the results of this preliminary study, it seems likely that Halstead values derived from a WO diagram may be used to predict those of the program to be written from the diagram.