

AN EXPERIMENT IN THE IMPLEMENTATION AND
APPLICATION OF SOFTWARE COMPLEXITY MEASURES

by

RANDALL ROBERT MEALS

B. S., Iowa State University, 1977

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

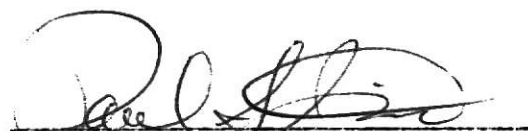
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1981

Approved by:


Major Professor

SPEC
COLL
LD
2668
.R4
1981
M42
c. 2

TABLE OF CONTENTS

	PAGE
CHAPTER 1	
Introduction.....	1
1.1 Purpose of Report.....	2
1.2 Content of Report.....	2
CHAPTER 2	
Software Complexity Measures.....	3
2.1 McCabe's Cyclomatic Complexity Measure.....	6
2.2 Halstead's Metrics.....	12
CHAPTER 3	
Tools to Measure Software Complexity.....	16
3.1 McCabe's Complexity Measure Evaluation Tool.....	18
3.2 Halstead's Metrics Evaluation Tool.....	21
3.3 Design Effort For McClure's Measure.....	24
CHAPTER 4	
Experience Applying Software Complexity Measures.....	25
4.1 Results of Applying McCabe's Measure.....	27
4.2 Results of Applying Halstead's Metrics.....	36
4.3 Interpreting Results.....	39
CHAPTER 5	
Summary of Experiment.....	41
5.1 Future Improvements and Directions.....	42
5.2 Conclusion.....	44
BIBLIOGRAPHY	
APPENDICES	
APPENDIX A	
F-Spec for McCabe's Tool	
1.0 Introduction.....	A-1
1.1 Scope.....	A-1
1.2 General Description.....	A-1
1.3 References.....	A-1
2.0 Environment.....	A-2
3.0 Computability Requirements.....	A-2
4.0 Performance Requirements.....	A-2
5.0 Interfaces.....	A-3
6.0 Functional Description.....	A-4
6.1 Functional Limitations.....	A-4
6.2 Description of Input.....	A-5
6.3 Description of Output.....	A-5
6.3.1 Listing of Output.....	A-5
6.3.2 Messages.....	A-6
6.3.2.1 Insufficient Space in SIRE-FILE.....	A-6
6.3.2.2 Insufficient Space in NAME-FILE.....	A-6

6.2.3.2	Search For Non-existent Condition- Name.....	A-7
6.2.3.4	Error Encountered During Search of SIRE-FILE.....	A-7
6.2.3.5	Error Encountered During Read of NAME-FILE Record.....	A-7
7.0	Reliability and Maintainability.....	A-8
7.1	Reliability.....	A-8
7.2	Maintainability.....	A-8

APPENDIX B

F-Spec For Halstead's Tool

1.0	Introduction.....	B-1
1.1	Scope.....	B-1
1.2	General Description.....	B-1
1.3	References.....	B-1
2.0	Environment.....	B-3
3.0	Compatability Requirements.....	B-3
4.0	Performance Requirements.....	B-3
5.0	Interfaces.....	B-4
6.0	Functional Description.....	B-5
6.1	Functional Limitations.....	B-6
6.2	Description of Input.....	B-6
6.3	Description of Output.....	B-7
6.3.1	Listing of Output.....	B-7
6.3.2	Messages.....	B-8
6.3.2.1	Error Encountered During Search of LINK-FILE.....	B-8
6.3.2.2	Error Encountered During Search of OPERAND-FILE.....	B-8
6.3.2.3	Insufficient Space in LINK-FILE.....	B-9
6.3.2.4	Insufficient Space in OPERAND-FILE.....	B-9
6.3.2.5	Error Encountered During Rewrite of LINK-FILE Record.....	B-9
6.3.2.6	Error Encountered During Rewrite of OPERAND-FILE Record.....	B-10
7.0	Reliability and Maintainability.....	B-11
7.1	Reliability.....	B-11
7.2	Maintainability.....	B-11

APPENDIX C

I-Spec For McCabe's Tool

1.0	Overview.....	C-1
1.1	Scope.....	C-1
1.2	General Description.....	C-1
1.3	References.....	C-1
2.0	Environment.....	C-2
3.0	Design Summary.....	C-2
3.1	System Flow.....	C-2
3.2	Logic Flow.....	C-3
3.3	Program Flow.....	C-4
4.0	Component Description.....	C-6
4.1	Local Data.....	C-6
4.2	Routines.....	C-10

5.0	Development and Maintenance.....	C-13
5.1	Development Systems.....	C-13
5.2	Maintenance Considerations.....	C-13
APPENDIX D		
I-Spec For Halstead's Tool		
1.0	Overview.....	D-1
1.1	Scope.....	D-1
1.2	General Description.....	D-1
1.3	References.....	D-1
2.0	Environment.....	D-3
3.0	Design Summary.....	D-4
3.1	System Flow.....	D-5
3.2	Logic Flow.....	D-5
3.3	Program Flow.....	D-6
4.0	Component Description.....	D-8
4.1	Local Data.....	D-8
4.2	Routines.....	D-13
5.0	Development and Maintenance.....	D-16
5.1	Development Systems.....	D-16
5.2	Maintenance Considerations.....	D-16
APPENDIX E		
McCabe's Tool Source Code		
APPENDIX F		
Halstead's Tool Source Code		
APPENDIX G		
McCabe's Measure - Tables of Results		
APPENDIX H		
Halstead's Metrics - Program A - Version 1		
APPENDIX I		
Halstead's Metrics - Program A - Version 2		
APPENDIX J		
Halstead's Metrics - Program B - Version 1		
APPENDIX K		
Halstead's Metrics - Program B - Version 2		
APPENDIX L		
Halstead's Metrics - Program C - Version 1		
APPENDIX M		
Halstead's Metrics - Program C - Version 2		
APPENDIX N		
Halstead's Metrics - Rank Order Frequency of Operators		

LIST OF FIGURES

	PAGE
Figure 2.1 Proposed Complexity Measures.....	4
Figure 2.2 Example of Flow Graph where $V(G) = 4$	7
Figure 2.3 Example of Flow Graph where $V(G) = 7$	9
Figure 2.4 Example of Flow Graph where $V(G) = 5$	11
Figure 4.1 Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program A.....	30
Figure 4.2 Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program B.....	31
Figure 4.3 Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program C.....	32
Figure 4.4 Frequency of Occurrence of Cyclomatic Complexity Values for the First Versions of the Measured Programs.....	33
Figure 4.5 Frequency of Occurrence of Cyclomatic Complexity Values for the Second Versions of the Measured Programs.....	34
Figure 4.6 Summary of Results of Applying McCabe's Cyclomatic Complexity Measure.....	35
Figure 4.7 Comparison of Actual Error Count of Estimated Error Count.....	38
Figure C.1 General Program Flow Through McCabe's Complexity Measure Evaluation Tool.....	C-5
Figure D.1 General Program Flow Through Halstead's Metrics Evaluation Tool.....	D-7
Figure G.1 Table of Cyclomatic Complexity Values Found in the First Version of Program A.....	G-1
Figure G.2 Table of Cyclomatic Complexity Values Found in the Second Version of Program A.....	G-2

Figure G.3	
Table of Cyclomatic Complexity Values Found in the First Version of Program B.....	G-3
Figure G.4	
Table of Cyclomatic Complexity Values Found in the Second Version of Program B.....	G-4
Figure G.5	
Table of Cyclomatic Complexity Values Found in the First Version of Program C.....	G-5
Figure G.6	
Table of Cyclomatic Complexity Values Found in the Second Version of Program C.....	G-6
Figure N.1	
Graph of Logarithmic Plot of Rank Order Frequency of Operators for the First Version of Program A.....	N-1
Figure N.2	
Graph of Logarithmic Plot of Rank Order Frequency of Operators for the Second Version of Program A.....	N-2
Figure N.3	
Graph of Logarithmic Plot of Rank Order Frequency of Operators for the First Version of Program B.....	N-3
Figure N.4	
Graph of Logarithmic Plot of Rank Order Frequency of Operators for the Second Version of Program B.....	N-4
Figure N.5	
Graph of Logarithmic Plot of Rank Order Frequency of Operators for the First Version of Program C.....	N-5
Figure N.6	
Graph of Logarithmic Plot of Rank Order Frequency of Operators for the Second Version of Program C.....	N-6

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Dave Gustafson, for his guidance and encouragement, to thank NCR Corporation for the availability of its resources, and to especially thank my wife, Linda, for her patience, understanding, support, and love.

CHAPTER 1

Introduction

This report is a summary of a project that focused on gaining experience in the implementation and application of software complexity measures. Much of the emphasis in adopting structured programming techniques is due to the perceived need to provide guidelines through which to reduce the complexity and increase the understandability of modules of software. A major factor motivating this emphasis is the high level of maintenance costs often experienced during the life-cycle of a piece of software. Intuitively, the complexity of a software module is somehow related to the difficulty experienced in developing, understanding, testing, modifying, and consequently maintaining that piece of software. With maintenance costs consuming the vast majority of the D.P. budgets of many organizations, there is a very real and pressing need to have some means by which to quantitatively measure software to see if it conforms to programming style guidelines which enforce and reinforce adherence to whatever structured programming techniques an organization selects. Software complexity measures are an attempt to fill this need.

The two complexity measures selected for this project are based on work done by Thomas J. McCabe [McCabe, 1976] and the late Maurice H. Halstead [Halstead, 1977].

1.1 Purpose of Report

This report is to serve as the documentation of the experiment with software complexity measures. It is by no means complete in the pursuit of this purpose. The goal of the project was to evaluate a set of software, for which the error history is known, against several different complexity measures. Many authors have presented what they view as a suitable means of measuring software. Some of these measures have even achieved a degree of universal acceptance, such as the ones selected for this project. An important aspect of this project is to determine if software which is indicated to be of high complexity in terms of a specific complexity measure proves to be problematic in terms of error history.

1.2 Content of Report

Presented in this report is the various aspects and issues involved in the experiment in the utilization of software complexity measures. Examined, to various degrees, are the selection of appropriate complexity measures, the history, determination, and implications of the selected complexity measures, and the experience of implementing and using tools to apply the selected software measures. There is an appendix containing functional and implementation specifications for the selected complexity measures, and listings of the tools used to apply the two measures used in this project, as well as listings and tables of the results of the software complexity measures applied to the set of software examined.

CHAPTER 2

Software Complexity Measures

Prior to the evolution and acceptance of Halstead's metrics and McCabe's cyclomatic measure, attempts at quantifying and limiting the complexity of a piece of software consisted of arbitrary guidelines which recognize that there is a rough correlation between module sizes and the understandability, reliability, and maintainability of those modules. The arbitrary guidelines took such forms as IBM limiting the size of modules to 50 source lines and TRW setting an upper bound of 2 pages of source code [McCabe, 1976]. While these limitations are no doubt useful to some extent, they are far from adequate when seeking a rigorous means by which to measure such aspects of software as clarity, testability, expected error rates, and goodness of the modularity scheme.

In 1972 Halstead published his theory of Software Physics [Halstead, 1972], which has since been renamed Software Science. Since then, there has been an avalanche of proposed software complexity measures and enhancements to proposed complexity measures. Although it is certainly not complete, in [Zolnowski and Simmons, 1980] the authors make a reasonable attempt at presenting a table listing, categorizing, and describing many of the software complexity measures proposed up to the point of publication of their report. The table is reproduced in Figure 2.1 for the reader's edification. Of these proposed measures, consideration for automation was given to Halstead's, McCabe's, McClure's, and Zolnowski and Simons', among others.

At this point, it is important to clarify the difference between measures of computational complexity of software and measures of the

Orientation of Measure	Description	Author
Control Flow	Cyclomatic Number	McCabe [1976]
	Count of Program Paths	Sullivan [1973]
	Enhancement of Cyclomatic Number (includes a count of logical conditions)	Meyers [1977]
	Measurement by the Pair (Cyclomatic Number, Operator Count)	Hansen [1978]
	Number of Multiple Entry Loops	Peterson [1973]
	Number of Knots	Woodward, et al. [1979]
	Cyclomatic Complexity Interval Plus # lines into/out of line of code	Cobb [1978]
Module Interaction	Number of Modules or Subsystems	Gilb [1977]
	$R \left(\frac{\text{Number of Module Linkages}}{\text{Number of Modules}} \right)$	Gilb [1977]
	Measure based on control structures and control variables	McClure [1978]
Data Reference	Measure of difficulty in understanding software's function based on components of sets on input and output	Chapin [1979]
Program Control	Minimal Intersection Number	Chen [1978]
Logical Complexity	$R \left(\frac{\text{Number of non-normal exits from a decision statement}}{\text{Total number of instructions}} \right)$	Gilb [1977]
Software Science	Metrics of software science predict complexity of a program	Hairstead [1977]
	Approach complexity via statistical (natural) language	Shooman & Laemmel [1977]
Composite Measure of Complexity	Index of Complexity based on Structure/Interaction/Instruction Mix/Data Reference Program Characteristics	Zolnoswski & Simmons [1977]
	Interface complexity/Computational complexity/I/O complexity/Readability	Thayer [1976]

Figure 2.1 Proposed Complexity Measures

psychological complexity of software. Computational complexity is understood as "the quantitative aspects of the solutions to computational problems" [Rabin, 1977]. An example of usage of computational complexity measures would be in comparing the efficiency of alternate algorithmic solutions [Curtis, et al., 1979]. Psychological complexity measures, of the type considered and utilized in this project, assess human performance in programming activities. Although there are certainly correlations between the two concepts, there is not expected to be any simple relationship between computational and psychological complexity measures.

The measures selected, those of T. McCabe and M. Halstead, are psychological software complexity measures. These two measures were selected out of the many choices due to their relative simplicity in what parameters are examined in the software being evaluated. This simplicity of parameters leads to a low level of computational complexity. When compared to psychological complexity measures proposed by McClure and Zolnowski and Simons, for instance, the cyclomatic complexity measure of McCabe and the Software Science metrics of Halstead are relatively easy to automate, yet the results of these measures satisfy many aspects of our intuitive notion of software complexity.

2.1 McCabe's Cyclomatic Complexity Measure

The definition of software complexity proposed by T. McCabe is based on the decision or control flow structures present in a piece of software. The actual metric used is the classical graph theory cyclomatic number indicating the number of regions in a graph. The graph used is the one classically known as the program control graph [Legard and Marcotty, 1975], where each node in the graph corresponds to a block of code where the flow is sequential and each of the arcs of the graph correspond to conditional branches in the blocks of code. The cyclomatic number of the graph is equal to the number of linearly independent control paths comprising a program [Curtis, et al., 1979]. Fundamentally, the measure then is the minimal number of independent paths in a module that has a single entry and a single exit.

The following examples will present three different, but equivalent methods by which McCabe's cyclomatic complexity measure may be computed. The first formula that can be used is:

$$V(G) = e - n + 2p$$

where $V(G)$ = cyclomatic complexity of graph,

e = number of edges,

n = number of nodes, and

p = number of connected components in the graph

($p = 1$ for single entry/single exit modules).

In Figure 2.2, the graph given there has 9 edges and 7 nodes. Thus

$$V(G) = e - n + 2$$

$$V(G) = 9 - 7 + 2$$

$$V(G) = 4$$

This means that there are four independent paths in the graph in

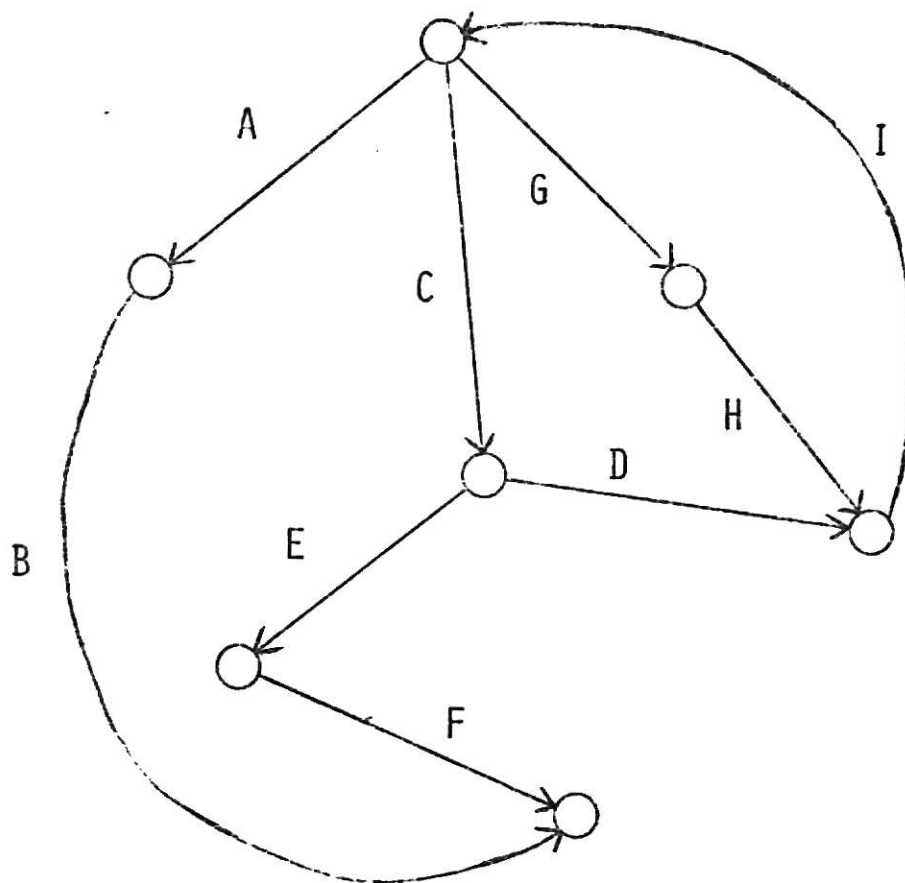


Figure 2.2 Example of Flow Graph where $V(G) = 4$

Figure 2.2 from which all paths can be made. These four independent paths are:

- (1) AB
- (2) GHI
- (3) CDI
- (4) CEF

A second formula for determining McCabe's cyclomatic complexity measure is:

$$V(G) = \pi + 1$$

where $V(G)$ = cyclomatic complexity of graph, and

π = number of predicate nodes (nodes with multi-exit paths).

In this formula, each binary exit node contributes 1 to the cyclomatic complexity, while each case-type node contributes one less than the number of exit-paths. In Figure 2.3, the graph given has three binary nodes and one case node. Thus

$$V(G) = \pi + 1$$

$$V(G) = [3 + (4 - 1)] + 1$$

$$V(G) = 7$$

Note that in this second example, it is not actually necessary to look at the flow graph. Instead, all that is necessary to compute the complexity of a module is to simply count the number of predicates (i.e. conditionals) in the code.

A third method for calculating McCabe's cyclomatic complexity measure is to use Euler's formula. If the flow graph is a planar graph, then:

$$V(G) = R$$

where R = the number of bounded regions on the graph, with the

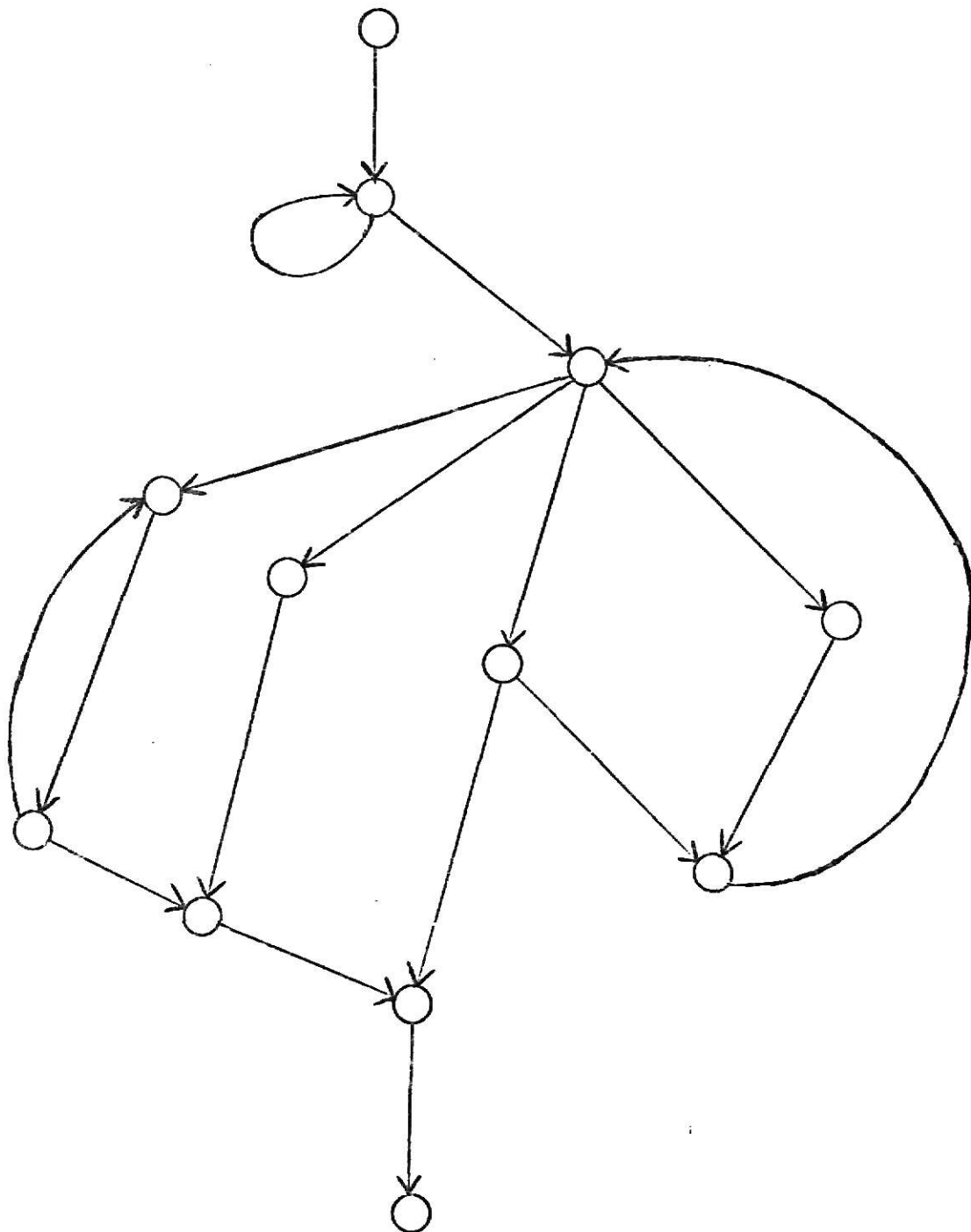


Figure 2.3 Example of Flow Graph where $V(G) = 7$

region external to the graph also counted as one.

In Figure 2.4, the graph has the five bounded regions identified.

By utilizing the three methods given for each of the graphs in Figure 2.2, Figure 2.3, and Figure 2.4, it can be seen that the methods are equivalent. A proof of the equivalence can be found in [McCabe, 1976].

There are several properties of McCabe's cyclomatic complexity measure, which are restated below from [McCabe, 1976]:

- 1) $V(G) \geq 1$.
- 2) $V(G)$ is the maximum number of linear independent paths in G ; it is the size of a basis set.
- 3) Inserting or deleting functional statements to G does not affect $V(G)$.
- 4) G has only one path if and only if $V(G) = 1$.
- 5) Inserting a new edge in G increases $V(G)$ by unity.
- 6) $V(G)$ depends only on the decision structure of G .

Although the concept was not used in this experiment, essential complexity will be briefly introduced here. Using the work of Rao Kasaraju [Kasaraju, 1974], McCabe expands on the concept of reducibility of structured and unstructured software, and defines essential complexity. The essential complexity, ev , of a module is the complexity that the flow graph can be reduced to when appropriate single entry/single exit subgraphs are successively replaced by structured single nodes. The minimal essential complexity of an unstructured program is 3, whereas a structured program is reducible to a program whose complexity is 1.

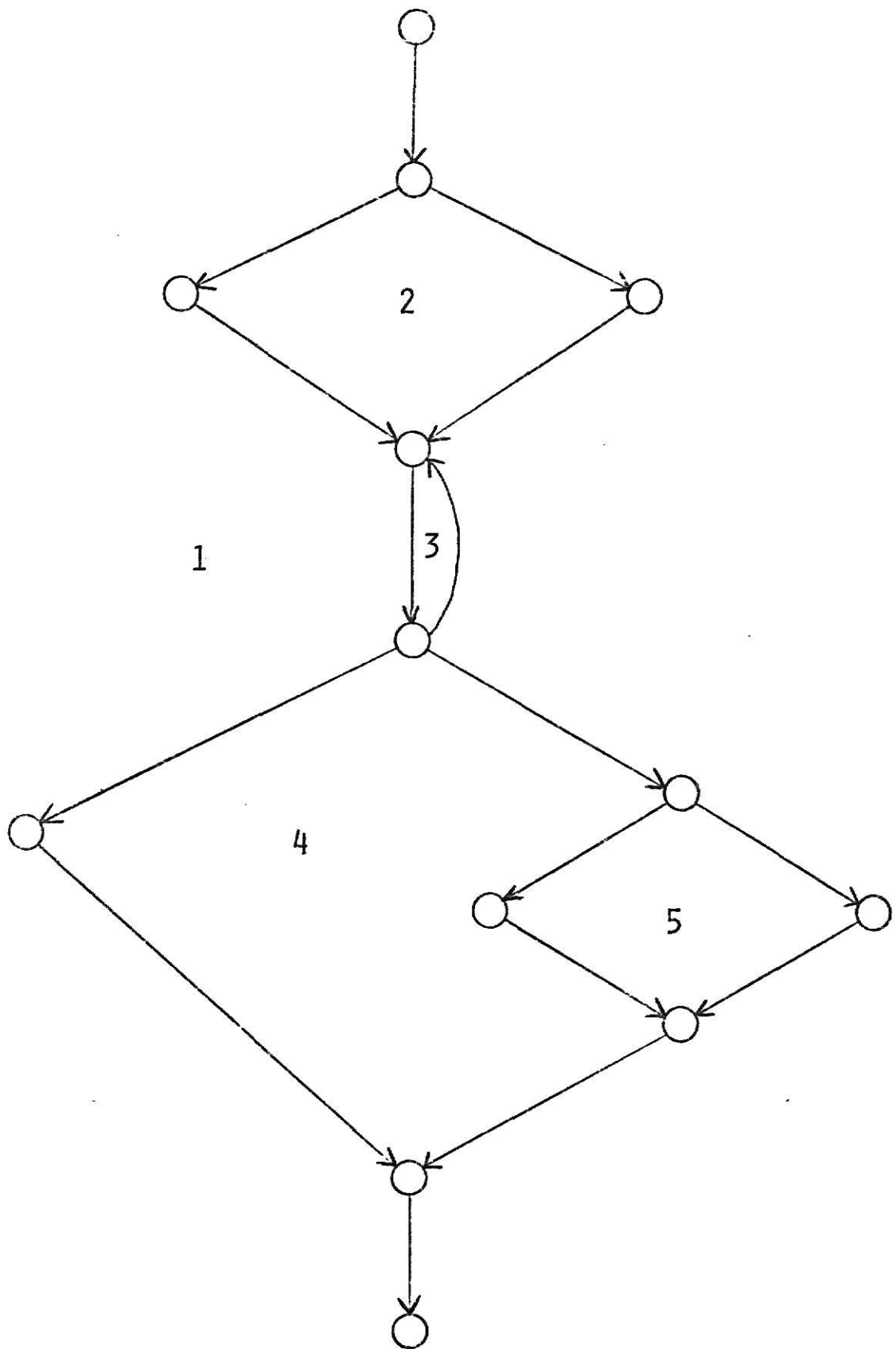


Figure 2.4 Example of Flow Graph where $V(G) = 5$

2.2 Halstead's Metrics

The metrics of Halstead's Software Science span more than just the simple measurement of the complexity of computer programs. The "software" of Software Science is any communication that appears in symbolic form in conformance with the grammatical rules of any language [Halstead, 1979]. The properties of a program, as defined by Halstead, are based upon four measured parameters:

- 1) η_1 - the number of unique operators,
- 2) η_2 - the number of unique operands,
- 3) N_1 - the total frequency of operators, and
- 4) N_2 - the total frequency of operands.

From these measured parameters, there are several properties that the theories of Software Science define as useful measures of a piece of software. The computed measures used in this project include:

- 1) Program Vocabulary - $\eta = \eta_1 + \eta_2$
- 2) Program Length - $N = N_1 + N_2$
- 3) Estimated Program Length - $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- 4) Program Volume - $V = (N_1 + N_2) \log_2 (\eta_1 + \eta_2)$
- 5) Implementation Level - $L = V^*/V$
- 6) Estimated Implementation Level - $\hat{L} = (2/\eta_1)(\eta_2/N_2)$
- 7) Estimated Potential Volume - $\hat{V}^* = \hat{L}V$
- 8) Effort Metric - $E_c = V/L$
- 9) Programming Time - $\hat{T} = E_c/S$, where S is Stroud's number, 18
- 10) Language Level - $\lambda = (L)^2 V$

Although a brief discussion of each of these computed measures will be given here, the reader is referred to [Halstead, 1979] for a more complete description of the fundamentals and implications of each metric.

The first two metrics in the list, Program Vocabulary, n , and Program Length, N , should almost appear obvious. The count of items in a program's vocabulary is simply the sum of the number of unique operators and the number of unique operands. The total number of occurrences of operators and operands is defined as the length of the program.

The equation for the estimation of Program Length, $\hat{N}$, is a result of an unexpected, and initially counterintuitive finding in the field of Software Science, that the lengths of programs are determined almost exclusively by the sizes of the vocabulary components n_1 and n_2 . The equation estimating Program Length is thus an expression of the vocabulary-length relationship. There have been studies [Elshoff, 1978] where correlation coefficients between N and $\hat{N}$ of over 0.98 have been reported for large groups of programs. There have been identified at least six "Impurity Cases" which can be the cause of a program not conforming to the vocabulary-length relation [Halstead, 1979]:

- (1) Complementary Operations, such as adding and immediately subtracting one operand from another,
- (2) Ambiguous Operands, the use of one operand name to serve more than one purpose, most likely in different parts of a program,
- (3) Synonymous Operands, the use of two different variables whose values are always the same,
- (4) Common Subexpressions, the failure to assign a name to the result of a frequently used calculation,
- (5) Unwarranted Assignment, the assignment of an operand name to

the result of a calculation used only one, and

- (6) Unfactored Expression, the failure to factor a factorable expression.

It is suggested that the inclusion of Impurity Cases in a piece of software makes that software harder to implement and understand.

The equation of Program Volume, V , is independent of the actual number of letters or characters in the identifiers found in a piece of software. The Program Volume is the fewest number of binary digits, or bits, with which a particular piece of software (program) can be represented.

The Implementation Level, L , of a piece of software is a measure of the brevity and compactness of an implementation of an algorithm or function. The Implementation Level may differ for functionally equivalent programs, depending on the power or level of language in which they are expressed. The equation estimating Implementation Level, $\hat{L}$, has been found to be close enough to be used interchangeably with L .

It is obvious that the volume of a program varies with the language in which the program is written. Translation of a piece of software into another language may result in an increase or a decrease in volume. Assuming that programs are translatable into ever more powerful languages, there would exist a potentially optimal language, and a potential volume, where the potential volume is the function of the program expressed as a procedure call in the potentially optimal language. Thus, the computed Potential Volume, V^* , represents the minimum possible volume associated with the function of a piece of software, and an absolute value against which others can be compared. For the purposes of this project, as L can be reasonably estimated, using the definition of L , V^* can be

estimated by substitution.

The Effort Metric, E_c , is a ratio of the Program Volume to the Implementation Level, and is an estimation of the mental effort required to create a program. The Program Volume metric serves as an indicator of the number of mental comparisons required to generate a program. The Implementation Level metric is the inverse of difficulty, as it is an indicator of the number of elementary mental discriminations (e. m. d.) required to make one average mental comparison. Thus, the difficulty of creating a program increases as the Program Volume increases, and decreases as the Implementation Level decreases, and the Program Volume multiplied by the inverse of the Implementation Level results in the total number of elementary mental discriminations required to generate the complete program.

By knowing the rate at which elementary mental discriminations occur, the amount of time required to generate the completed program can be computed. Fortunately, psychology provides such a rate, which is nearly constant, and does not vary significantly with intelligence. The number is named in honor of John Stroud, who did work in the examination of psychological time [Stroud, 1966], and who first reported this rate, which, for time in seconds, is taken as 18 [Halstead, 1979].

The equation for determining the Language Level, λ , of a program is equivalent to the Implementation Level multiplied by the Potential Volume. It has been found experimentally that as Potential Volume is increased, Implementation Level will decrease proportionately [Halstead, 1979]. Thus, the average Language Level for programs written in the same language tends to remain constant over a wide range of program sizes. For English, the average Language Level is 2.16.

Chapter 3

Tools to Measure Software Complexity

The automation of the measures of Halstead and McCabe was accomplished by writing separate programs for each measure in COBOL. The machines on which the measures were developed are members of NCR's I-Family of computing systems. The I-Family consists of members of the NCR 8000 series of computers that operate in the interactive direct processing or "I" mode. The I-Family of systems is oriented to conversational, online processing where there is continual interaction between machine and operator. The members of the I-Family range from desk-top micros to minis to mainframes. The major development system used in this project was an I8270 with 1024K bytes of memory, and the IMOS V operating system.

The COBOL language was selected due to its commonality among the I-Family systems. This commonality extends to a common COBOL virtual machine into which COBOL programs are compiled, using a common COBOL compiler. The COBOL virtual machine is emulated via various hardware/firmware/software mixtures in the individual systems of the I-Family. The common COBOL compiler and virtual machine thus allow the complexity measures to be transportable in both source and object form between the members of the I-Family. An additional advantage of using COBOL as the implementation language is that, with minor modifications to remove I-Family specific screen-formatting instructions, the source for the complexity measure tools is transportable to NCR's virtual, or "V" mode systems. The "V" mode systems use the VRX operating system, and have

a separate COBOL machine for the VRX environment. The VRX environment is implemented on small through medium through large mainframes and is primarily batch oriented.

3.1 McCabe's Complexity Measure Evaluation Tool

The tool to evaluate a COBOL program in terms of McCabe's cyclomatic complexity measure was the first of the two complexity measure tools written. A functional specification for the tool can be found in Appendix A, an implementation specification for the tool can be found in Appendix C, and a source listing of the completed COBOL program implementation can be found in Appendix E.

The initial problem to be solved in the design of the McCabe's Complexity Measure Evaluation Tool was the evolution of a technique to isolate tokens from the source program being evaluated. The tool first had to handle the fact that the COBOL source input could be in either fixed- or variable-length record format. The code in the Declaratives section of the program does this by handling fixed-length records as the standard situation and variable-length records as the exception case. The token isolation routines, when presented with a line of code, will sequentially scan each character of the line, and isolate source words based on known delimiters. If a literal is encountered, a special routine which can accomodate embedded quotation marks and allow literals up to the standard maximum of 255 characters is invoked.

In the calculation of McCabe's cyclomatic complexity measure, the methodology was to count all the decision points in the Procedure Division of a program, summing the count for each paragraph and section in the program, and for the whole program, and calculating the average complexity of the paragraphs in the entire program. In all cases, a paragraph is assumed to be equivalent to a module and thus have an initial complexity value of one. In a complex conditional expression, each simple conditional expression was assigned a value of one. In COBOL, a conditional

expression may consist of a condition-name or a combination of condition-names, as well as explicit conditional expressions. These condition-names can be user-defined, such as switches and 88-level items, or part of the language definition, such as "ALPHABETIC", "NUMERIC", etc. In order to recognize user-defined condition-names that may occur in conditional expressions, and thus include their implicit complexity values in the computations, the Environment Division is scanned for any 88-level items. In the case of an 88-level item, a single value is counted as adding a value of one and a range, such as "A THRU Z", is counted as adding a value of two. This is due to the assumption that the single value can be tested using an equality conditional and the range can be tested using greater-than-or-equals and less-than-or-equals conditionals.

In order to retain the information associated with condition-names, specifically switches and 88-level items, two scratch work files are used. The reason for this is that the language that the tool is written in, COBOL, does not have dynamic data structures, such as the heap in PASCAL, and the work files are the only reasonable means by which to handle information of indeterminate size. The side effect of this is, given that the source program is coded as efficiently as possible, the majority of the execution time is spent in disk accesses, mostly due to obtaining the next source line, but sometimes due to searching for a particular condition-name.

In the implementation phase, the largest problem encountered was smoothing out the token isolation and advancement procedures, especially the parsing of specific COBOL syntactic structures in the Procedure Division. In order to facilitate this debugging process,

code was added to the McCabe's Complexity Measure Tool so that if the tool is compiled with the Debug option on the system execute command, and then executed, useful information will be displayed on the CRT device. This information includes the current source word and source line being scanned, the current line number, the current running total cyclomatic complexity for the current module, and a pointer to the last character in the current source line that has been scanned. While this information is very useful, there is a performance penalty to be paid in the time consumed displaying all this information to the CRT device. A rough estimate of the penalty would be in the range of a 5:1 ratio. The optional code to do these displays is indicated by a D in column 7 of the source, and if the Debug option is not specified during the invoking of a compilation, the COBOL compiler treats these lines as comments.

3.2 Halstead's Metrics Evaluation Tool

The tool to evaluate a COBOL program in terms of the selected metrics from Halstead's Software Science was developed after the McCabe's Complexity Measure Evaluation Tool was designed, implemented, and tested. A functional specification for the tool can be found in Appendix B, an implementation specification for the tool can be found in Appendix D, and a source listing of the completed COBOL program implementation can be found in Appendix F.

As the tool for evaluating a COBOL program in terms of McCabe's cyclomatic complexity measure had already been designed, implemented, and reasonably verified, the design and coding effort for the tool for Halstead's metrics was made much easier. This is due to the ability to borrow large portions of code to handle the task of the isolation of tokens in the source program being examined. The design effort then centered on the collection of operators and operands, and their frequencies, and the calculation of the individual metrics.

Fundamental to the computation of Halstead's metrics for a given COBOL program is the determination of exactly what constitutes an operator or operand. In trying to do so, one soon discovers that once past the general definitions presented in Software Science, it becomes a seat-of-the-pants affair. For the purpose of this project, only the Procedure Division was scanned for operators and operands, with operators being predetermined from the set of COBOL reserved words that satisfy the author's notion of operators. Along the way, the decision was made to count each pair of parentheses as a single occurrence of a parentheses-class operator and to associate double-quotation marks surrounding literals as an integral part of those literals. Delimiters and separators, such

as periods, commas, and semicolons, which are optional in many circumstances, are not counted as either operators or operands.

As a point of interest, to calculate the base two logarithms required in some of the formulas for the metrics (see Section 2.2), a power-series method was used to obtain the natural logarithm of a given number, with a conversion factor applied to obtain the base two logarithm. The exact technique is described in [Lyusternik, Chervonenkis, and Yanpol'skii, 1965].

In the original pass at implementing a tool to evaluate Halstead's Software Science metrics, only one scratch work file was used. This file served as a dynamic data structure which held the individual operands' id-names found in the source input, along with a count of their frequency of occurrence. When an operand is encountered, a search of the list of existing operands is made to determine whether the operand has already been encountered, and thus exists in the list, or if a new entry needs to be made. As the operands were entered into the list in the order of first occurrence in the source input, the search of the list was originally implemented in a sequential fashion. It soon became obvious that it was possible to improve upon this. The tool was modified so that there are now two scratch work files. One of the files still holds the operands' id-names and frequency counts in order of first occurrence in the source input. The other file is a set of links, chaining together all operands with the same first alphabetic character, unless the first character is non-alphabetic, for which case a separate chain of non-alphabetic characters exists in the file. A performance improvement during the scanning of an operand results, as only the chain of entries with the same first character, or the chain of non-alphabetic characters,

needs to be searched. The improvement is in the range of a 10:1 decrease in execution time required for a 10,000 line COBOL source program.

As with the tool to evaluate McCabe's complexity measure, code was added so that if the tool is compiled with the Debug option on the system execute command, and then executed, useful information will be displayed to the CRT device. This information includes the current source word and source line being scanned, the current line number, and a pointer to the last character in the current source line that has been scanned. There is, as with the other tool, a performance penalty to be paid when executing in this mode, due to the time consumed in displaying this information to the CRT device. A rough estimate of the penalty is similarly in the 5:1 range. In both cases, the optional code to do these displays is indicated by a "D" in column 7 of the source; these lines are treated as comments if the Debug option is not specified during the invoking of a compilation.

3.3 Design Effort For McClure's Measure

In the course of selecting the complexity measures to be used in this project, serious consideration was given to using the complexity measure proposed by Carma McClure [McClure, 1978]. It was observed that the results of applying McClure's measure appear to be very meaningful and useful. Based on this observation, an effort was made to design a software tool to automate the evaluation of a piece of software in terms of McClure's measure.

Once most of the design was complete, and some preliminary throw-away code written, it became apparent that there would be severe performance problems with an implementation of McClure's measure. Essentially, the difficulty lies in that the measure requires that variables be evaluated in terms of their relationship to the execution hierarchy of a program. Two passes through the source are made; the first to save a representation of the execution hierarchy, and the control variables directly involved, and the second to examine the source for instances of each of the control variables in terms of modules where they are referenced, modules where they are modified, and modules where they are not referenced or modified, but are part of the execution hierarchy. The examination of each variable's ancestor-dependent relationships against the above-mentioned categories would consume an inordinate amount of execution time, not to mention a large amount of dynamic data structures (files in a COBOL implementation), and be unacceptable as a tool to be regularly used in the development and maintenance of COBOL source programs. Few programmers would have the patience to wait days for the results of evaluating a 10,000 line program.

CHAPTER 4

Experience Applying Software Complexity Measures

For this project, a set of software for which the error history is known was evaluated using the complexity measure tools developed. The set of programs examined was not selected for any special reason, other than they were available and have an associated error history. As is to be expected, the error history is not a complete record of all past and future errors in the selected software, but serves as a snapshot from which to detect tendencies and trends. The measured software is a set of three utility programs, written in COBOL, where all were written by different programmers, in different circumstances, and were passed through the hands of several different individuals during the development, maintenance, and enhancement phases.

A review of the error history, or snapshot of the error history of the three programs examined, reveals reported problem counts of 32, 11, and 5. For convenience, we will call the program with 32 reported problems Program A, the program with 11 reported problems Program B, and the program with 5 reported problems Program C. As can be seen, Program A has been fairly error-prone in comparison to the other two programs.

A description of the development environment of the utility programs that were examined assists in understanding the results of this experiment. The programs have had two distinct development phases. The first phase led up to the initial version, or quiet point, in the development cycle. The second phase involved the correction of reported problems

and perceived functional design shortcomings, including a new feature added to the set of utilities. The new feature resides in Program B, with new code to interface to the feature existing in Program A. Program A has had a total of six different programmers working on it; five on the first version, and two on the second version, with one programmer overlapping the two phases. Program A's development has been very chaotic, as can be seen by the high personnel changeover. In the two versions Program A grew from 4956 lines of code to 5948 lines. Program B had two different programmers involved in developing its first version and a third programmer developed the second version. From the first to the second version Program B grew from 4239 to 5001 lines of code. Program C experienced a comparatively stable development environment, with the same programmer responsible for both the first and the second versions. Program C is the largest of the three programs, and grew from 9416 to 9425 lines between the two versions.

Subjectively rating the programs against accepted structured programming techniques, Program A appears to be the least structured, and Program C appears to be the most structured, with Program B falling somewhere in between the two.

4.1 Results of Applying McCabe's Measure

Applying the cyclomatic complexity measure of McCabe to the utility programs examined reveals some interesting results. Tables containing the various cyclomatic complexity values found and the number of modules that had each value are presented in Appendix G. Figure 4.1, Figure 4.2, and Figure 4.3 show the change in the frequency of occurrence of cyclomatic complexity values between the first and second versions of each of the measured programs. The contrast of the frequency of occurrence of cyclomatic complexity values between the three measured programs for their first versions is shown in Figure 4.4, and for their second versions is shown in Figure 4.5. As can be seen in a summary of some of the various aspects of the evaluation in Figure 4.6, Program C, the program with the fewest problem reports, has the highest value for the average complexity of the paragraphs in the program. Unlike Program A and Program B, Program C had a decrease in the number of decision points between the two versions. It can also be seen that for the program with the highest number of problem reports, Program A, the average complexity of the paragraphs in the program increased from the first to the second version of the program. The increase is in contrast to the other two programs, in which the average complexity of the programs' paragraphs decreased.

In Program A, the number of paragraphs with a cyclomatic complexity value of greater than 10 increased from 9 to 14 from the first version to the second. In the first version the highest value is 18; in the second it is 24. Out of the 12 sections in the program that underwent change in complexity values in the first version, the average complexity of the paragraphs in the sections increased in 9 of the 12 sections.

In Program B, the number of paragraphs with a complexity value greater than 10 was 6 in both the first and second versions. In the first version the highest value is 13 and is found in two paragraphs; in the second version the highest value is 16. There were 10 sections in the first version that underwent change in complexity values, and out of those 10 sections, only 3 resulted in sections with an increase in complexity.

In comparison, Program C had only 3 sections that had a change from the first to the second version of the program which resulted in a change in complexity values. Out of the three sections, only one had an increase in the average complexity of its paragraphs. In Program C, there are 20 paragraphs with a complexity value greater than 10 in both the first and second versions. The highest value in both is 27, with two paragraphs in each having that value. There are also two paragraphs in each with a value of 23, one with 19, and two with 17. This program, of the three, has the highest individual complexity values and the highest average. Program C is also the program with the lowest number of problem reports.

A further evaluation of the results of applying McCabe's measure was conducted by a visual inspection of the source code of the first and second versions of the three utility programs. The evaluation consisted of comparing the problem reports with the areas of code associated with a problem report, and showed that the problem areas of code are not necessarily simply the paragraphs where the cyclomatic complexity exceeds a value of 10. Rather, it appears that the problem areas are those where the average complexity of groups of related paragraphs is comparatively high. This is probably an indication of the amount of

mental effort required to understand the interaction between the affected paragraphs. For a further discussion of the visual inspection of code, see Section 4.3.

-

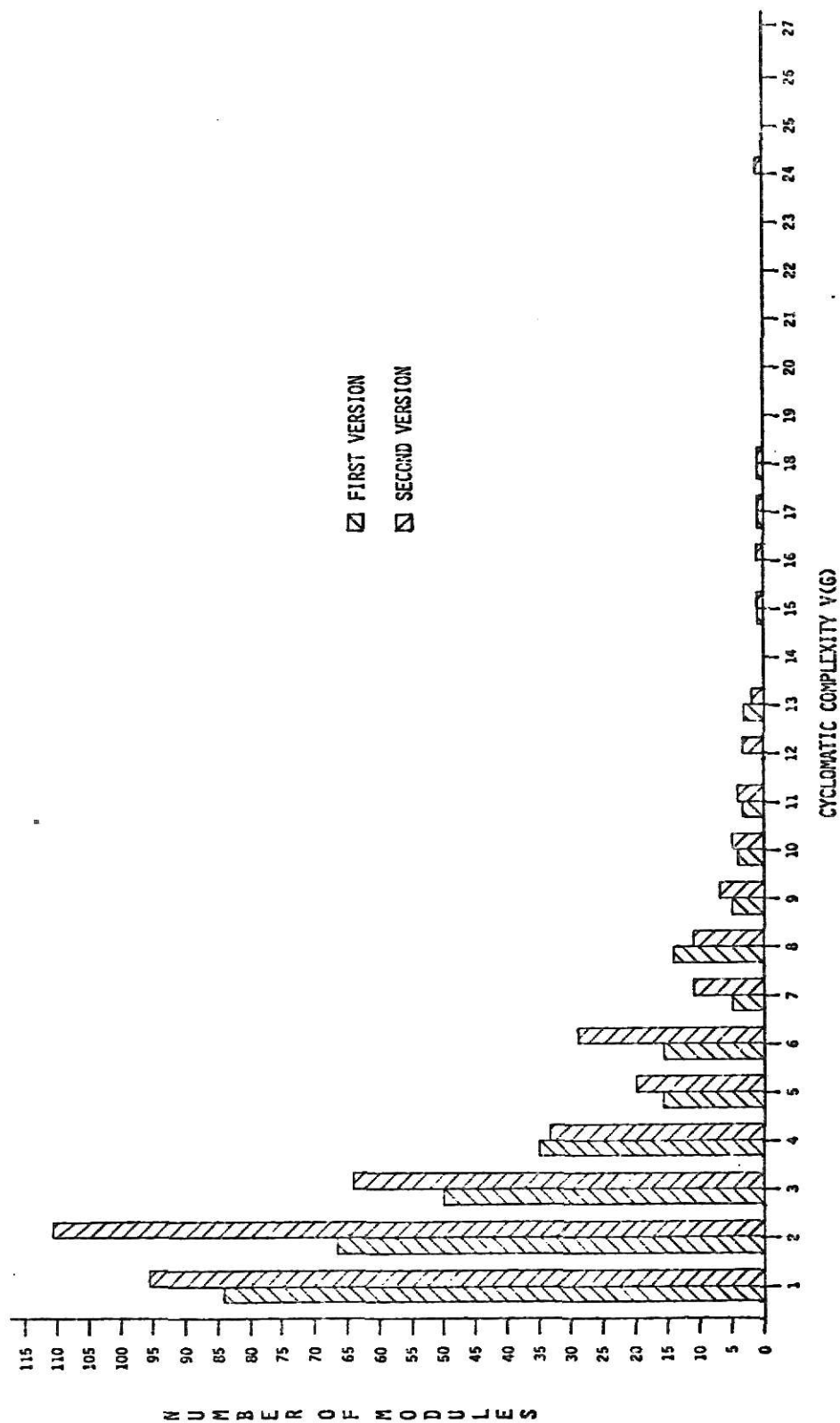


Figure 4.1 Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program A.

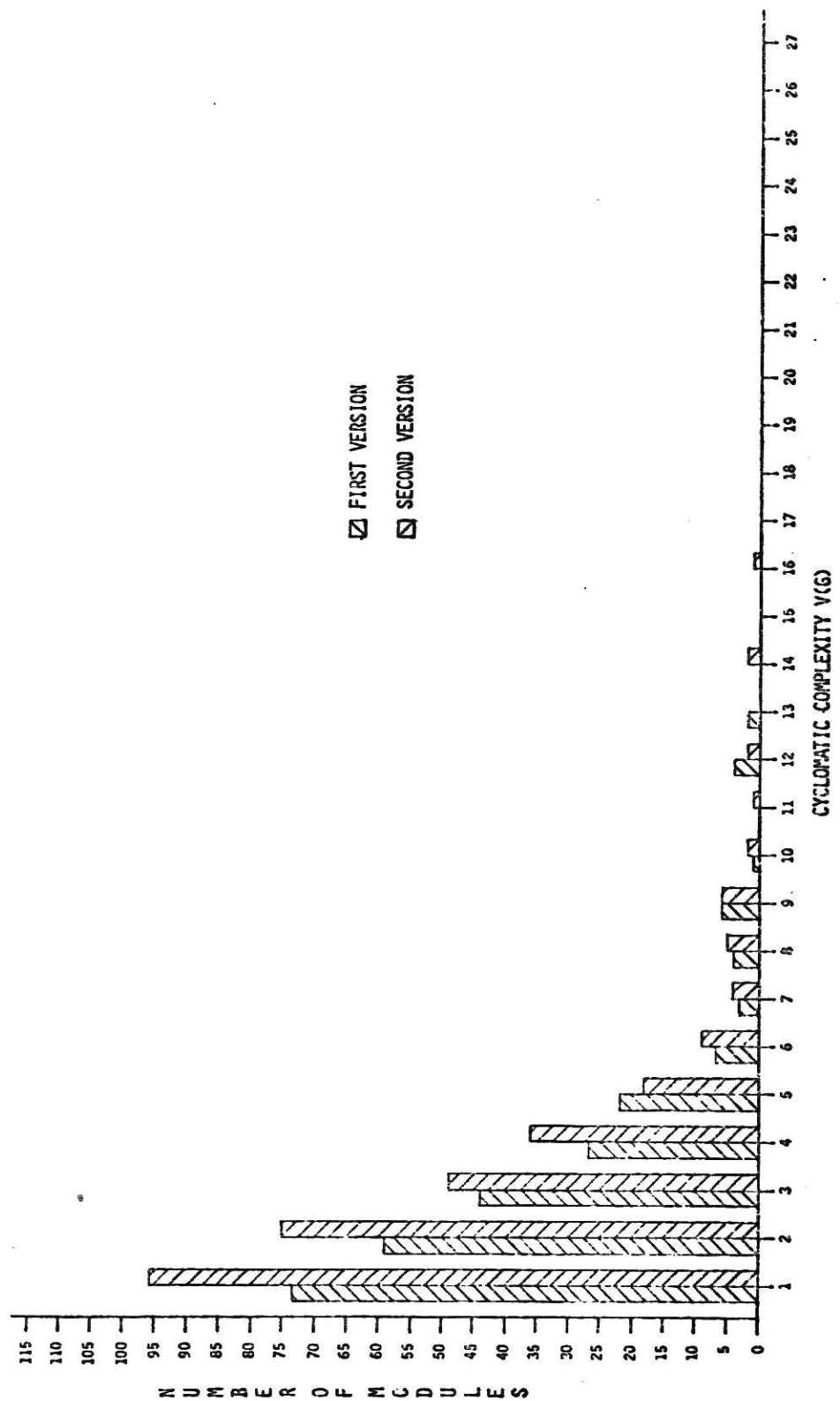


Figure 4.2 Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program B.

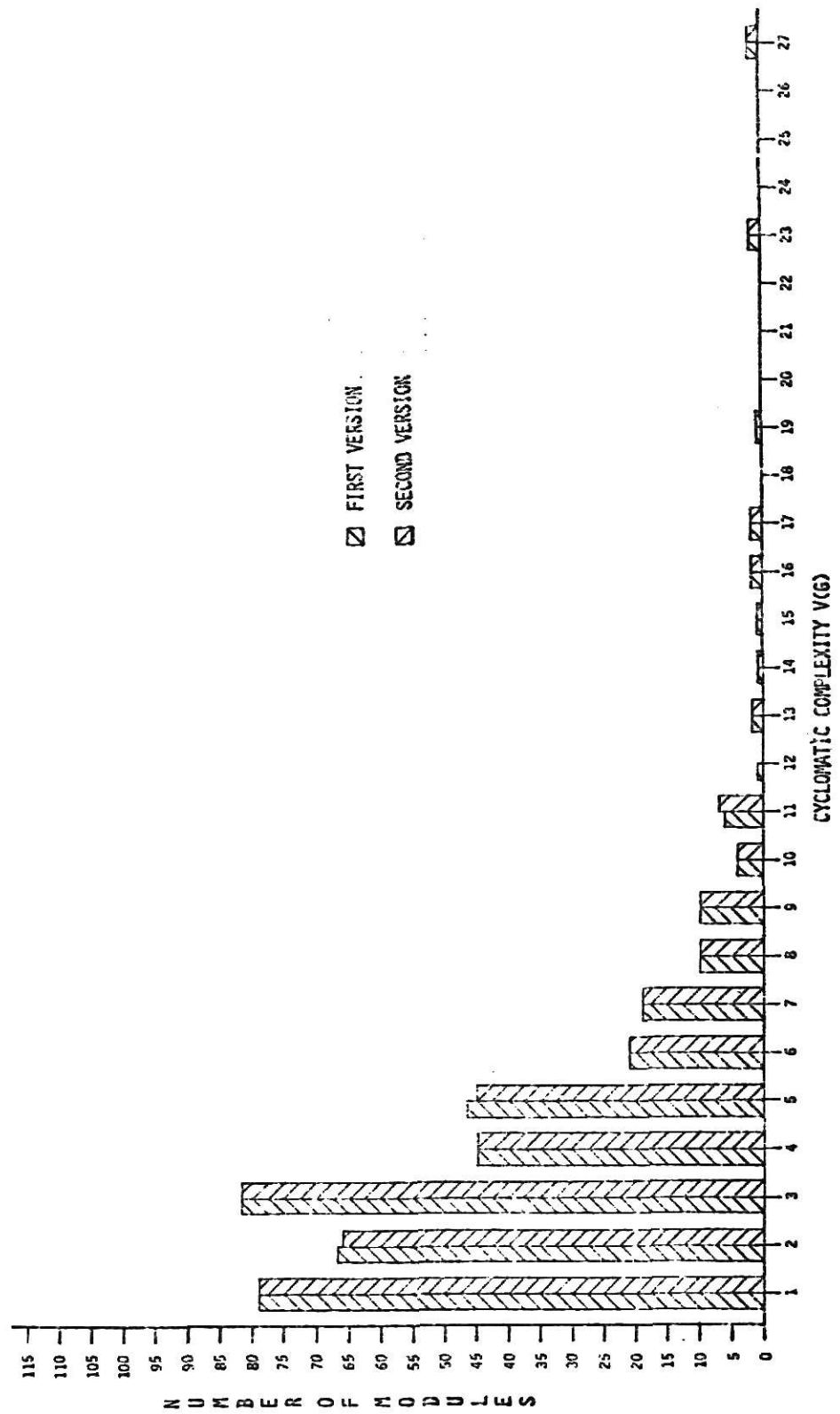


Figure 4.3 Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program C.

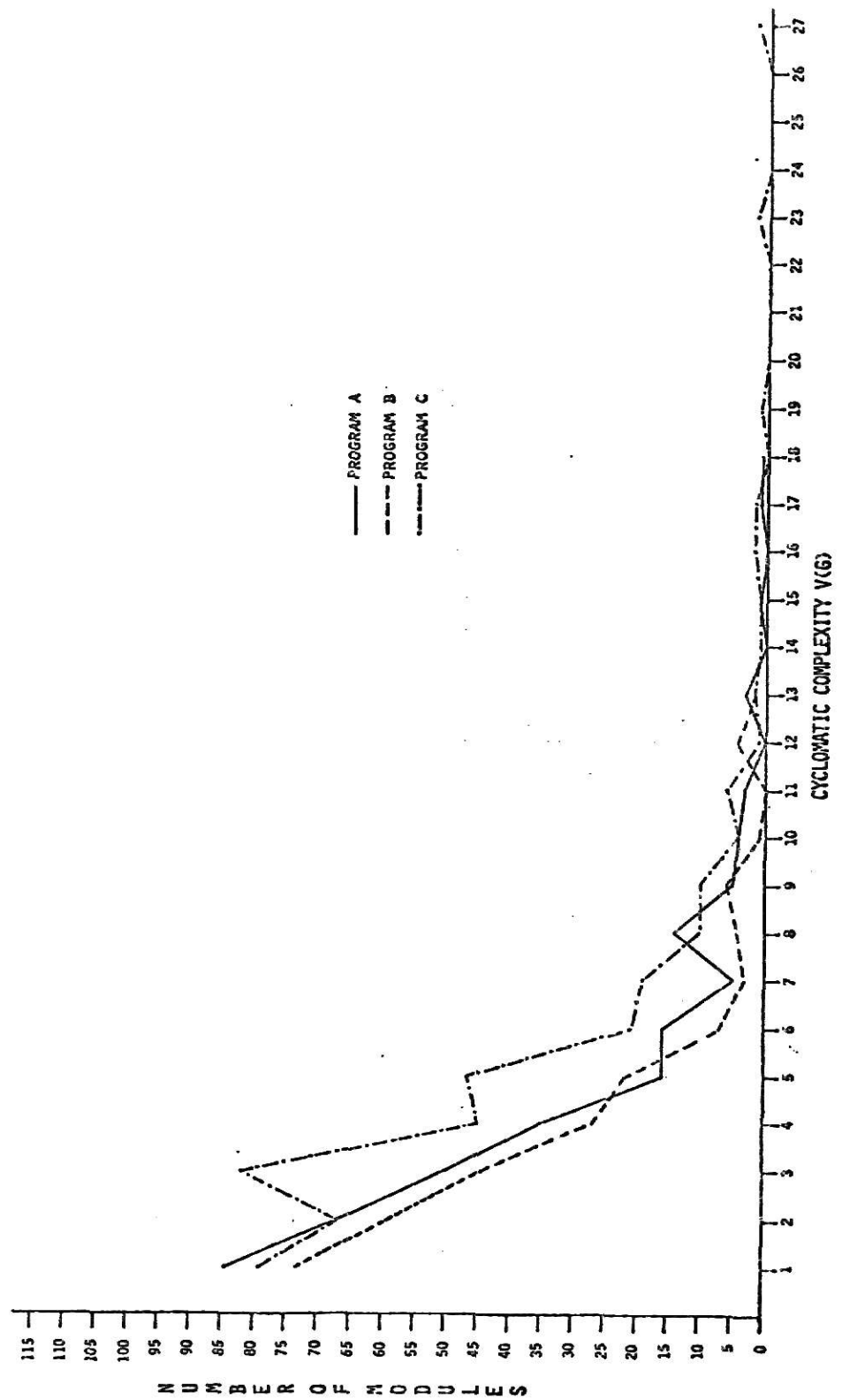


Figure 4.4 Frequency of Occurrence of Cyclomatic Complexity Values for the First Versions of the Measured Programs.

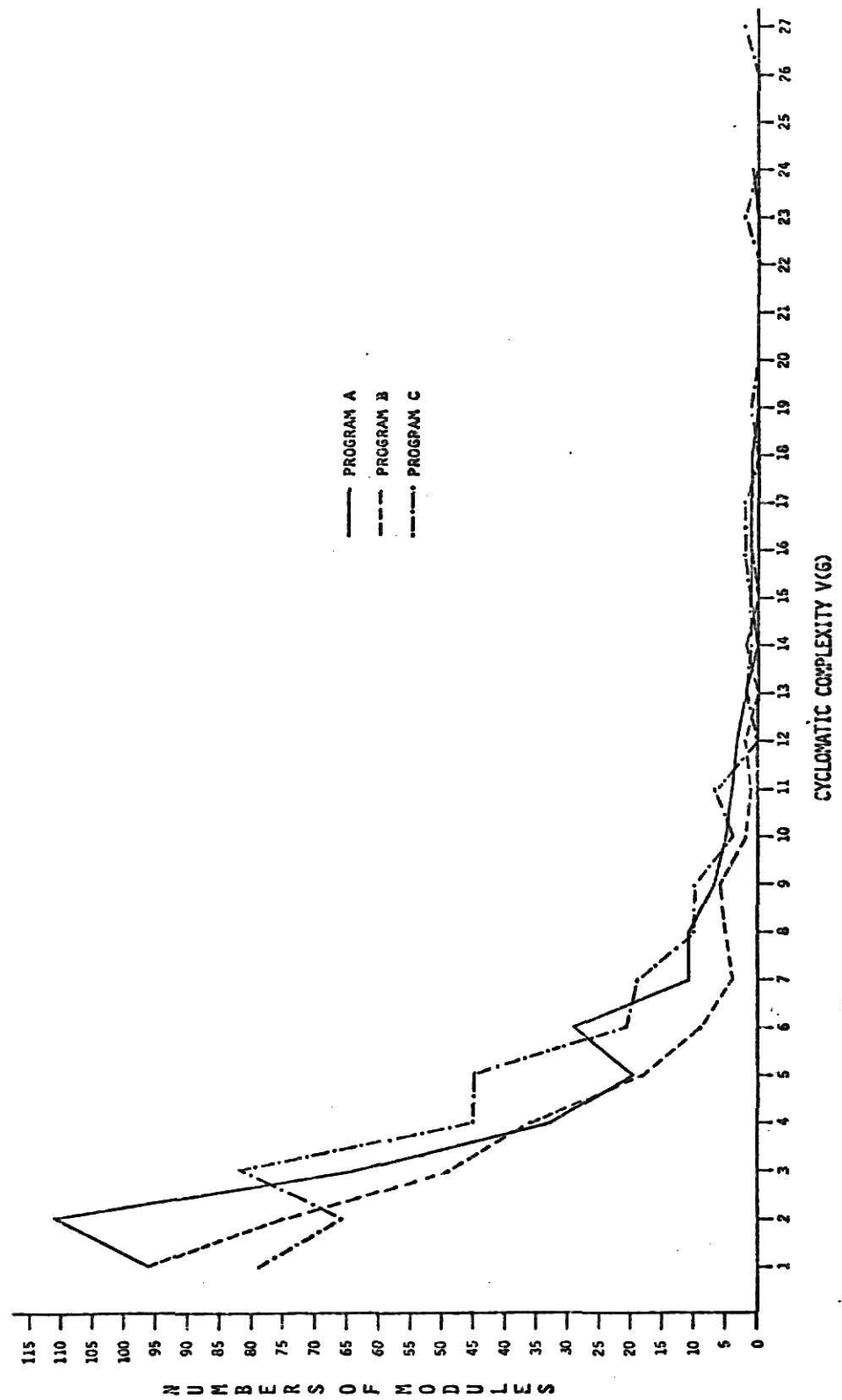


Figure 4.5 Frequency of Occurrence of Cyclomatic Complexity Values for the Second Versions of the Measured Programs.

	Version	Number of Problem Reports	Total # of Modules	Total # of Decision Points	Average Cyclomatic Complexity Value of Modules
Program A	1		304	1037	3.4112
	2	32	383	1362	3.5561
Program B	1		252	744	3.0714
	2	11	307	904	2.9446
Program C	1		404	1662	4.1139
	2	5	401	1648	4.1097

Figure 4.6 Summary of Results of Applying McCabe's Cyclomatic Complexity Measure.

4.2 Results of Applying Halstead's Metrics

The results of applying the metrics of Halstead's Software Science are shown in Appendix H through Appendix M, with the ordered listing by frequency of operands not shown, for brevity. It can be seen that Program C, the program with the fewest error reports, has the lowest Implementation Level and the lowest Language Level. Interestingly, Program C also has considerably larger values for the Effort Metric and for the Program Volume than do Program A and Program B.

L. M. Ottenstein [Ottenstein, 1979], using psychological evidence, has derived a formula for predicting an expected number of problem reports, using the implementation level of a program as quantitative measure of the repetition of elementary mental discriminations. The formula

$$\hat{B}_V = LE/E_0$$

is equivalent to

$$\hat{B}_V = V/E_0,$$

where $\hat{B}_V$ = estimation of number of problem reports,

L = Implementation Level,

E = Number of elementary discriminations that could be in error, and

E_0 = number of correct elementary mental discriminations to be expected before the first erroneous one.

Ignoring the language level of individual programs, and assuming that the average language level utilized by the programmer is the same as that used in natural language communication, which is 2.16, Ottenstein shows that $E_0 = 3000$. Applying this to both versions of the three programs examined yields the values in Figure 4.7. There appears to be a cor-

relation between the predicted amount of error reports, $\hat{B}_V$, and the actual trend in the error history, indicated by the number of problem reports, B_V , for Program A and Program B. However, there is a large discrepancy in the value for $\hat{B}_V$ and B_V for Program C. For further discussion and interpretation of the results of applying the metrics of Halstead's Software Science see Section 4.3, which follows this section.

It has been observed that a pattern exists in the rank ordered frequency distributions of operators in computer programs. Essentially, the pattern is a reflection of the distributions following a logarithmic function. Graphs of the logarithmic values of the frequency of occurrence of the three measured programs can be found in Appendix N. For further discussion of the concept, see [Zweben and Halstead, 1979].

	Actual Error Count B_v	Estimated Error Count $\hat{B}_v$ for Version 1	Estimated Error Count $\hat{B}_v$ for Version 2
Program A	32	40.01	52.83
Program B	11	23.60	29.76
Program C	5	74.98	75.84

Figure 4.7 Comparison of Actual Error Count to Estimated Error Count

4.3 Interpreting Results

It is obvious that if one's view of the expected error history of the three utility programs examined was to be derived only from the results of Halstead's and McCabe's complexity measures, that Program A and Program B behave close to expectations, where Program C is a counter-example. There are a number of possible explanations for this phenomenon. It is possible that Program C is indeed a software time-bomb, full of errors, that simply has not been tested adequately. However, in this case, it appears that all three programs have had equally rigorous testing.

The more likely situation is that due to the relatively stable development environment of Program C, the programmer who developed both versions of the program, and who probably writes code with few errors in the first place, was able to take full advantage of the learning curve due to his experience with the software, and thus had an easier time debugging any erroneous situations encountered. What can be interpreted from the predicted high error rate for Program C is that when other programmers become responsible for its maintenance and enhancement, if they are any less familiar with the original design and intentions, a very difficult time moving along the learning curve will be experienced (For a discussion of the learning curve concept, see [Webb, 1976]). Due to this, modifications introduced will have a high chance of being incorrect and/or causing undesirable side effects. In short, Program C will probably be much more difficult to maintain and enhance than will Program A and Program B.

As mentioned in Section 4.1, a visual inspection of the source code for the two versions of the three utility programs that were measured

was conducted. Subjectively interpreting the results of that inspection, Program A appears to have confused, ad hoc, coding in many places, which reflects the unstable development environment it experienced. Program C appears to be over-modularized, with the author failing to use common modules effectively, resulting in a more complicated control structure. This over-modularization is reflected in the Estimated Program Length for Program C being almost three times the measured Program Length, which is an indication of the presence of Impurity Cases. Such a large variation between the Estimated Program Length and the measured Program Length does not exist for Program A or Program B. Program B lies somewhere in between Program A and Program C, with most of its awkward coding due to rearranging code around system memory constraints. To summarize, after examining the results of the measures, the visual inspection of the source code revealed no surprises.

CHAPTER 5

Summary of Experiment

This project evolved into three major phases of activity. The first phase consisted of a survey of literature to become acquainted with the variety of psychological software complexity measures that have been proposed by various authors and to select the measures to be used based on appropriate criteria, especially ease of automation. Once this process was completed, the next step was to automate the selected measures. During this phase, it became apparent that the cyclomatic measure of McCabe and the metrics of Halstead's Software Science were acceptable, but the measure proposed by McClure was too complex in the computational sense. The design, coding, and testing/debugging of the tools to automate the evaluation of the measures of McCabe and Halstead was accompanied by preparation of documentation of the effort consisting of functional and implementation specifications.

The last phase of the project was to apply the measures against a set of software, in this case a set of utility programs with a known error history. The results of this evaluation verify the usefulness of the selected measures. The measures can indeed predict the error-prone nature of code, both in the present and future versions of programs.

5.1 Future Improvements and Directions

In any undertaking, there is always room for changes and, hopefully, improvements. This endeavor is no exception. During the course of this project, it was realized that there were some things which could, and perhaps should, be changed, if time had permitted (an elusive excuse at best.)

One alteration which would be immensely useful would be to alter the tools so that as they scan the COBOL source, they output to the CRT device an indication that they are alive and well. The best format for this is to output the name of the tool, followed by the source line number in increments of ten. Displaying source line numbers in increments of ten should not provide too severe of a performance degradation, yet still be frequent enough to serve the intended purpose.

Other alterations to be explored include those where tradeoffs would occur between performance and transportability or functionality. An example of this type of change would be to use the system-dependent sort utilities to improve the performance of such activities as the outputting of operators and operands by order of frequency by Halstead's Metrics Evaluation Tool. Or perhaps the bodies of code composing the modules could be extracted into a work file in a first pass, and then the work file sorted by frequency, then separated by known operators into operators and operands. Certainly there are more elegant solutions the author has not thought of nor seriously considered.

There are a multitude of directions future work stemming from this project may take. A reasonable next phase in the aftermath of this experiment is to use the tools developed to evaluate other groups of software. Indeed, this step has already occurred, and the programmers

who are using the tools on an informal basis in the industrial environment are enthusiastic about their availability. The author is proposing that in the future tools for the two measures for each programming language used be implemented and adopted as part of the standard development and maintenance process. The justification for proposing McCabe's cyclomatic complexity measure and Halstead's Software Science metrics as basis for standardization in the industrial programming environment derives in part from the results of this experiment and in part from the supportive evidence to be found in the literature cited in this report.

5.2 Conclusion

This exercise in the implementation and application of McCabe's cyclomatic complexity measure and the metrics selected from Halstead's Software Science has demonstrated that it is reasonable and useful to automate these measures via coding in COBOL to apply against COBOL source programs. Although they certainly do not serve as complete indicators of complexity, the selected measures do appear to serve as useful indicators of not only error-prone sections of code, but as warning flags for otherwise innocent appearing piece of software.

BIBLIOGRAPHY

- Baker, Albert L. and Stuart H. Zweben, "The Use of Software Science in Evaluating Modularity Concepts," IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 110 - 120.
- Berlinger, Eli, "An Information Theory Based Complexity Measure," Proceedings of the National Computer Conference, 1980, pp. 773 - 779.
- Chapin, Ned, "A Measure of Software Complexity," Proceedings of the National Computer Conference, 1979, pp. 995 - 1002.
- Chen, E. T., "Program Complexity and Programmer Productivity," IEEE Transactions on Software Engineering, Vol. SE-4, No. 3, May 1978, pp. 187 - 193.
- Cobb, G. W., "A Measurement of Structure For Unstructured Programming Languages," Proceedings of the Software Quality and Assurance Workshop, Nov. 1978, pp. 140 - 147.
- Comer, Douglas, and Maurice H. Halstead, "A Simple Experiment in Top-Down Design," IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 105 - 109.
- Curtis, Bill, et al., "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 96 - 104.
- Elshoff, J. L., "An Investigation into the Effect of the Counting Method Used on Software Science Measurements," ACM SIGPLAN Notices, Vol. 13, No. 2, pp. 30 - 45.
- Gilb, T., Software Metrics, Winthrop Publishers, Inc., 1977.
- Gordon, Ronald D., "A Qualitative Justification for a Measure of Program Clarity," IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 121 - 128.
- Gordon, Ronald D., "Measuring Improvements in Program Clarity," IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 79 - 90.
- Halstead, Maurice H., "Natural Laws Controlling Algorithm Structure?", ACM SIGPLAN Notices, Vol. 7, No. 2, 1972, pp. 19 - 26.
- Halstead, Maurice H., Elements of Software Science, Elsevier North-Holland, New York, 1977.

- Halstead, Maurice H., "Advances in Software Science," *Advances in Computers*, Vol. 18, Academic Press, Inc., New York, 1979, pp. 119 - 172.
- Hennell, M. A., W. M. McNicol, and J. Hawkins, "The Static Analysis of Cobol Programs," *ACM SIGSOFT, Software Engineering Notes*, Vol. 5, No. 4, Oct. 1980, pp. 17 - 25.
- Jensen, Randall W. and Charles C. Tonies, *Software Engineering*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.
- Kosaraju, Rao, "Analysis of Structured Programs," *J. Comput. Sys. Sci.*, Vol. 9, Dec. 1974, pp. 232 - 255.
- Legard, H., and M. Marcotty, "A Generalogy of Control Structures," *Comm. of the ACM*, Vol. 18, Nov. 1975, pp. 629 - 639.
- Lyusternik, L. A., O. A. Chervonenkis, and A. R. Yanpol'skii, *Handbook for Computing Elementary Functions*, Pergamon Press Ltd., Oxford, 1965.
- McCabe, Thomas J., "A Complexity Measure," *IEEE Transactions of Software Engineering*, Vol. SE-2, No. 4, Dec. 1976, pp. 308 - 320.
- McClure, Carma L., "A Model for Program Complexity Analysis," *Proceedings of the 3rd International Conference on Software Engineering*, 1978, pp. 149 - 157.
- McTap, John L., "The Complexity of an Individual Program," *Proceedings of the National Computer Conference*, 1980, pp. 767 - 771.
- Meyers, Glenford T., "An Extension to the Cyclomatic Measure of Program Complexity," *ACM SIGPLAN Notices*, Vol. 12, No. 10, Oct. 1977, pp. 61 - 64.
- Ottenstein, Linda M., "Quantitative Estimates of Debugging Requirements," *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 5, Sept. 1979, pp. 504 - 514.
- Peterson, W. W., T. Kasami, and N. Tokura, "On the Capabilities of While, Repeat, and Exit Statements," *Comm. of the ACM*, Vol. 16, Nov. 1973, pp. 503 - 512.
- Rabin, M. O., "Complexity of Computations," *Comm. of the ACM*, Vol. 20, No. 9, 1977, pp. 625 - 633.
- Shooman, M., and Laemmel, A., "Statistical Theory of Computer Programs - Information Content and Complexity," *Digest of Papers COMPCON Fall 1977*, pp. 341 - 347.
- Stroud, J. M., "The Fine Structure of Psychological Time," *Ann. N. Y. Acad. Sci.*, 1966, pp. 623 - 631.

- Sullivan, J. E., Measuring the Complexity of Computer Software, MITRE Corporation, MTR-2648, Vol. V, June 1973.
- Tanik, Murat M., "A Comparison of Program Complexity Prediction Models," ACM SIGSOFT Software Engineering Notes, Vol. 5, No. 4, Oct. 1980, pp. 10 - 12.
- Thayer, T. A., et al., Software Reliability Study, RADC-TR-76-238, Aug. 1976.
- Walsh, Thomas J., "A Software Reliability Study Using a Complexity Measure," Proceedings of the National Computer Conference, 1979, pp. 761 - 768.
- Webb, Samuel C., Managerial Economics, Houghton Mifflin Co., Boston, 1976.
- Woodfield, Scott N., "An Experiment on Unit Increase in Problem Complexity," IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 76 - 79.
- Woodward, M. R., et al., "A Measure of Control Flow Complexity in Program Text," IEEE Transactions on Software Engineering, Vol. SE-5, No. 1, Jan. 1979, pp. 45 - 50.
- Yau, Stephan S., and James S. Conllofello, "Some Stability Measures for Software Maintenance," IEEE Transactions on Software Engineering, Vol. SE-6, No. 6, Nov. 1980, pp. 545 - 552.
- Zolnowski, Jean M., and Dick B. Simmons, "Measuring Program Complexity," Digest of Papers Fall COMPCON 1977, pp. 336 - 340.
- Zolnowski, Jean M., and Dick B. Simmons, "Measuring Program Complexity in a COBOL Environment," Proceedings of the National Computer Conference, 1980, pp. 757 - 766.
- Zweben, Stuart H., and Maurice H. Halstead, "The Frequency Distribution of Operators in PL/1 Programs," IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 91 - 95.

APPENDIX A

F-Spec For McCabe's Tool

The following is the Functional Specification for the tool to evaluate McCabe's Cyclomatic Software Complexity Measure.

FUNCTIONAL SPECIFICATION
FOR
McCABE'S COMPLEXITY MEASURE EVALUATION TOOL

1.0 Introduction

1.1 Scope

This document defines the functional capabilities of the McCabe's Complexity Measure Evaluation Tool.

1.2 General Description

The McCabe's Complexity Measure Evaluation Tool, which operates on systems which support NCR I-Family COBOL, is designed to scan a COBOL source program in terms of McCabe's cyclomatic complexity measure, and output the results to a listing file. The files to be used by McCabe's Complexity Measure Evaluation Tool must be established while under control of the operating system.

1.3 References

The following references should be consulted for additional information on the concepts discussed in this specification:

- (1) T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. SE-2, pp. 308 - 320, Dec. 1976.
- (2) Meals, Randall R., Implementation Specification for McCabe's Complexity Measure Evaluation Tool. (See Appendix C.)

2.0 Environment

The McCabe's Complexity Measure Evaluation Tool is designed for use on any of the members of NCR's I-Family of computing systems. Currently, the hardware systems forming the I-Family include 8080-based, 605-based, and 8400-based user systems. The operating systems forming the I-Family environment include the IDPS, MDPS, IMOS III, IMOS V, and IRX operating systems. Execution of I-Family COBOL is also possible under the MCITS and TCOS systems.

3.0 Compatability Requirements

Compatability amongst the systems which support the I-Family COBOL execution environment is assured via the usage of I-Family COBOL as the implementation vehicle.

4.0 Performance Requirements

Although there are no actual execution time requirements, it is understood that to be of any use in the COBOL development and maintenance environment, the McCabe's Complexity Measure Evaluation Tool should perform in a reasonable and timely fashion, i.e. not take days to evaluate the complexity of a 10,000 line program.

5.0 Interfaces

The McCabe's Complexity Measure Evaluation Tool is invoked using a sequence of JCL commands that: assign the COBOL program source file and two scratch work files; assigns the printer or spool file so the output can be printed or saved; and executes the program.

```
ASSIGN I TO source-program  
ASSIGN NAME-FILE TO <work-file-1>,NEW,SCRATCH,nnnn  
ASSIGN SIRE-FILE TO <work-file-2>,NEW,SCRATCH,nnnn
```

The work files must be large enough to contain all the necessary information regarding the condition-names in the source program and their qualification hierarchy. A size of 1000 sectors for each of the work files generally provides sufficient space for a COBOL program of approximately 10,000 source lines.

6.0 Functional Description

The McCabe's Complexity Measure Evaluation Tool is designed as a utility for COBOL application developers. It provides a means of automating the calculation of McCabe's cyclomatic complexity measure for every paragraph and section, as well as averages for every section and an average and total for the entire program. The tool consists of a COBOL program, which accepts COBOL source as input, utilizes two scratch work files, and produces a listing output of the results of the measure applied to the source input.

6.1 Functional Limitations

The McCabe's Complexity Measure Evaluation Tool will not handle COPY files. If any condition-names occur in COPY files for the Environment Division or the Data Division, and the condition-names are referenced in the Procedure Division, an error message will be displayed to the CRT device upon the tool's failure to discover the condition-name in the scratch work file. All COPY statements in the Procedure Division will simply result in the complexity of the associated code in the COPY files not being included in the computation of cyclomatic complexity.

6.2 Description of Input

The source program processed by the McCabe's Complexity Measure Evaluation Tool must be in a file having one of the following formats:

1. Disk file with variable-length records varying in size from 1 to 80 characters, and a block size of 512 characters.
2. A cassette file with variable-length records varying in size from 1 to 80 characters, and a block size of 512 characters.
3. A card file (having fixed-length records of 80 characters).

6.3 Description of Output

6.3.1 Listing of Output

The listing output from McCabe's Complexity Measure Evaluation Tool consists of the individual paragraph names, section names, and section numbers found in the source program, with the paragraph names followed by their cyclomatic complexity, and the section names and section numbers followed by the total and average cyclomatic complexity of the paragraphs of the section. After the entire source program is scanned, the total and average cyclomatic complexity for the paragraphs of the program is output.

6.3.2 Messages

The following messages will be output to the CRT device if a detected situation warrants such a message to be displayed.

6.3.2.1 Insufficient Space in SIRE-FILE

The message

RAN OUT OF SIRE-FILE SPACE

indicates that an insufficient amount of space has been assigned to the SIRE-FILE for the particular COBOL program being examined. The solution is to restart the initiating procedure, assigning more sectors to the SIRE-FILE work file. An increment of 500 sectors would be a reasonable estimate.

6.3.2.2 Insufficient Space in NAME-FILE

The message

RAN OUT OF NAME-FILE SPACE

indicates that an insufficient amount of space has been assigned to the NAME-FILE for the particular COBOL program being examined. The solution is to restart the initiating procedure, assigning more sectors to the NAME-FILE work file. An increment of 500 sectors would be a reasonable estimate.

6.3.2.3 Search For Non-existent Condition-Name

The message

```
TRIED TO SEARCH FOR A NON-EXISTENT CONDITION-NAME <name>  
IN LINE NUMBER <line-number>
```

indicates one of three possible situations:

- (1) There was a physical error in searching the SIRE-FILE,
- (2) The condition-name searched for is defined in a portion of the Environment Division or Data Division contained in a COPY file, or
- (3) There is a logic error in the McCabe's Complexity Measure Evaluation Tool program.

6.3.2.4 Error Encountered During Search of SIRE-FILE

The message

```
SIRE-FILE SEARCH ERROR
```

probably indicates a physical I/O error occurred while trying to read a record in the SIRE-FILE work-file. The user should restart the initiating procedure.

6.3.2.5 Error Encountered During Read of NAME-FILE Record

The message

```
NAME-FILE READ ERROR
```

probably indicates a physical I/O error occurred while trying to read a record in the NAME-FILE work file. The user should restart the initiating procedure.

7.0 Reliability and Maintainability

7.1 Reliability

The McCabe's Complexity Measure Evaluation Tool should be robust enough to recover from situations created by references to condition-names defined in COPY files.

7.2 Maintainability

In order to facilitate maintainability, the COBOL source program for McCabe's Complexity Measure Evaluation Tool, when compiled using the Debug option of the system execute statement, and executed, will display additional information to the CRT device, including the source line and the line number of the source line being scanned.

APPENDIX B

F-Spec For Halstead's Tool

The following is the Functional Specification for the tool to evaluate Halstead's Software Science Metrics.

FUNCTIONAL SPECIFICATION
FOR
HALSTEAD'S METRICS EVALUATION TOOL

1.0 Introduction

1.1 Scope

This document defines the functional capabilities of the Halstead's Metrics Evaluation Tool.

1.2 General Description

The Halstead's Metrics Evaluation Tool, which operates on systems which support NCR I-Family COBOL, is designed to scan a COBOL source program in terms of the Software Science metrics defined by M. Halstead, and output the results to a listing file. The files to be used by the Halstead's Metrics Evaluation Tool must be established while under control of the operating system.

1.3 References

The following references should be consulted for additional information on the concepts discussed in this specification:

- (1) Meals, Randall R., Implementation Specification for Halstead's Metrics Evaluation Tool. (See Appendix D.)
- (2) Halstead, M. H., "Advances in Software Science," Advances in Computers, Vol. 18, Academic Press, Inc., New York, 1979.

- (3) Lyusternik, L. A., O. A. Chervonenkis, and A. R. Yanpol'skii, Handbook for Computing Elementary Functions, Pergamon Press Ltd., Oxford, 1965.

2.0 Environment

The Halstead's Metrics Evaluation Tool is designed for use on any of the members of NCR's I-Family of computing systems. Currently, the hardware systems forming the I-Family include 8080-based, 605-based, and 8400-based user systems. The operating systems forming the I-Family environment include the IDPS, MDPS, IMOS III, IMOS V, and IRX operating systems. Execution of I-Family COBOL is also possible under the MCITS and TCOS systems.

3.0 Compatability Requirements

Compatability amongst the systems which support the I-Family COBOL execution environment is assured via the usage of I-Family COBOL as the implementation vehicle.

4.0 Performance Requirements

Although there are no actual execution time requirements, it is understood that to be of any use in the COBOL development and maintenance environment, the Halstead's Metrics Evaluation Tool should perform in a reasonable and timely fashion, i.e. not take days to evaluate a 10,000 line program.

5.0 Interfaces

The Halstead's Metrics Evaluation Tool is invoked using a sequence of JCL commands that: assign the COBOL program source file and two scratch work files; assigns the printer or spool file so the output can be printed or saved; and executes the program.

```
ASSIGN I TO <source-program>  
ASSIGN OPERAND TO <work-file-1>,NEW,SCRATCH,nnnn  
ASSIGN LINK TO <work-file-2>,NEW,SCRATCH,nnnn  
EXECUTE OHALSTEAD
```

The work files must be large enough to contain all the necessary information regarding the operands in the source program and link lists chaining them together. A size of 1000 sectors for each of the work files generally provides sufficient space for a COBOL program of approximately 10,000 source lines.

6.0 Functional Description

The Halstead's Metrics Evaluation Tool is designed as a utility for COBOL application developers. It provides a means of automating the calculation of the metrics of Halstead's Software Science discipline. The tool consists of a COBOL program, which accepts COBOL source as input, utilizes two scratch work files, and produces a listing output of the results of the measures applied to the source input. The following metrics from Software Science are utilized in the evaluation of the source input:

- 1) The number of unique operators
- 2) The number of unique operands
- 3) The total frequency of operators
- 4) The total frequency of operands
- 5) Program Vocabulary
- 6) Program Length
- 7) Estimated Program Length
- 8) Program Volume
- 9) Implementation Level
- 10) Potential Volume
- 11) Effort Metric
- 12) Programming Time
- 13) Language Level

For further explanation of the above metrics, see reference (2) in Section 1.3 of this document.

6.1 Functional Limitations

The Halstead's Metrics Evaluation Tool will not handle COPY files. All COPY statements in the Procedure Division of the COBOL source input will simply result in the properties of the associated code in the COPY files not being accounted for in the computation of the specific metrics.

6.2 Description of Input

The source program processed by the Halstead's Metrics Evaluation Tool must be in a file having one of the following formats:

1. Disk file with variable-length records varying in size from 1 to 80 characters, and a block size of 512 characters.
2. A cassette file with variable-length records varying in size from 1 to 80 characters, and a block size of 512 characters.
3. A card file (having fixed-length records of 80 characters).

6.3 Description of Output

6.3.1 Listing of Output

The listing output from Halstead's Metrics Evaluation Tool consists of a one page summary of the results of the evaluation of the source input program in terms of the specific metrics, followed by separate lists of the individual operators and the individual operands, and their frequency of occurrence in the COBOL source program, in order of their frequency of occurrence.

6.3.2 Messages

The following messages will be output to the CRT device if a detected situation warrants such a message to be displayed.

6.3.2.1 Error Encountered During Search of LINK-FILE

The message

```
LINK-FILE SEARCH ERROR  
LINK KEY = <link-key>  
NUMBER OF LINKS = <number-of-links>
```

probably indicates a physical I/O error occurred while trying to read a record in the LINK work file. The user should restart the initiating procedure.

6.3.2.2 Error Encountered During Search of OPERAND-FILE

The message

```
OPERAND-FILE SEARCH ERROR  
OPERAND KEY = <operand-key>  
NUMBER OF UNIQUE OPERANDS = <number-of-unique-operands>
```

probably indicates a physical I/O error occurred while trying to read a record in the OPERAND work file. The user should restart the initiating procedure.

6.3.2.3 Insufficient Space in LINK-FILE

The message

```
RAN OUT OF SPACE IN LINK-FILE  
LINK KEY = <link-key>
```

indicates that an insufficient amount of space has been assigned to the LINK work file for the particular COBOL program being examined. The solution is to restart the initiating procedure, assigning more sectors to the LINK work file. An increment of 500 sectors would be a reasonable estimate.

6.3.2.4 Insufficient Space in OPERAND-FILE

The message

```
RAN OUT OF SPACE IN OPERAND-FILE  
OPERAND KEY = <operand-key>
```

indicates that an insufficient amount of space has been assigned to the OPERAND work file for the particular COBOL program being examined. The solution is to restart the initiating procedure, assigning more sectors to the OPERAND work file. An increment of 500 sectors would be a reasonable estimate.

6.3.2.5 Error Encountered During Rewrite of LINK-FILE Record

The message

```
PROBLEM REWRITING LINK-FILE  
LINK KEY = <link-key>
```

probably indicates a physical I/O error occurred while trying to rewrite a record in the LINK work file. The user should restart the initiating procedure.

6.3.2.6 Error Encountered During Rewrite of OPERAND-FILE Record

The message

```
PROBLEM REWRITING OPERAND-FILE  
OPERAND KEY = <operand-key>
```

probably indicates a physical I/O error occurred while trying to rewrite a record in the OPERAND work file. The user should restart the initiating procedure.

7.0 Reliability and Maintainability

7.1 Reliability

The Halstead's Metrics Evaluation Tool should be robust enough to recover from situations created by the existence of COPY statements in the Procedure Division of the COBOL program being scanned.

7.2 Maintainability

In order to facilitate maintainability, the COBOL source program for Halstead's Metrics Evaluation Tool, when compiled using the DEBUG option of the system execute statement, and executed, will display additional information on the CRT device, including the source line and the line number of the source line being scanned.

APPENDIX C

I-Spec For McCabe's Tool

The following is the Implementation Specification for the tool to evaluate McCabe's Cyclomatic Software Complexity Measure.

IMPLEMENTATION SPECIFICATION
FOR
McCABE'S COMPLEXITY MEASURE EVALUATION TOOL

1.0 Overview

1.1 Scope

This document specifies the implementation details of the McCabe's Complexity Measure Evaluation Tool.

1.2 General Description

The McCabe's Complexity Measure Evaluation Tool, which operates on systems which support NCR I-Family COBOL, is designed to scan a COBOL source program in terms of McCabe's cyclomatic complexity measure, and output the results to a listing file. The files to be used by McCabe's Complexity Measure Evaluation Tool must be established while under control of the operating system.

1.3 References

The following references should be consulted for additional information on the concepts discussed in this specification:

- (1) T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. SE-2, pp. 308 - 320, Dec. 1976.
- (2) Meals, Randall R., Functional Specification for McCabe's Complexity Measure Evaluation Tool. (See Appendix A.)

2.0 Environment

The McCabe's Complexity Measure Evaluation Tool is designed for use on any of the members of NCR's I-Family of computing systems. Currently, the hardware systems forming the I-Family include 8080-based, 605-based, and 8400-based user systems. The operating systems forming the I-Family environment include the IDPS, MDPS, IMOS III, IMOS V, and IRX operating systems. Execution of I-Family COBOL is also possible under the MCITS and TCOS systems.

3.0 Design Summary

The McCabe's Complexity Measure Evaluation Tool is designed as a utility for COBOL application developers. It provides a means of automating the calculation of McCabe's cyclomatic complexity measure for every paragraph and section, as well as averages for every section and an average and total for the entire program.

3.1 System Flow

The McCabe's Complexity Measure Evaluation Tool is invoked as a normally executable COBOL program via the appropriate system commands, with its input in logical file I and its output in logical file PRINT-FILE. There must also be assigned scratch work files associated with the logical files NAME-FILE and SIRE-FILE.

3.2 Logic Flow

The McCabe's Complexity Measure Evaluation Tool is a COBOL program. It accepts as input COBOL source. The tool scans the COBOL source input and first extracts the names and complexity of any switches in the Environment Division and any 88-level condition-names in the Data Division of the source program, along with the sire names of the qualification hierarchy of the condition-names, saving these collected names and associated information into scratch work files. The tool then scans the Procedure Division of the source program, looking for conditional expressions. If a conditional expression is found, the next source word is examined to see if it is a conditional operator. If it is not, the conditional expression is assumed to be a condition-name, and the condition-name work file is searched for a match. If no such match is found, an error is assumed, and reported via a message to the CRT device, and the scanning is resumed. If it is found in the condition-name work file, the implicit complexity of the condition-name is added to the running totals retained for the current paragraph, current section, and the entire program. For non-condition-name situations, each conditional operator contributes unity. All paragraphs have a minimum complexity of one if they are non-empty, and a complexity of zero if there are no lines of code. The scanning and computing occurs until the end of the source program.

3.3 Program Flow

The general program flow through the McCabe's Complexity Measure Evaluation Tool is shown in Figure C.1.

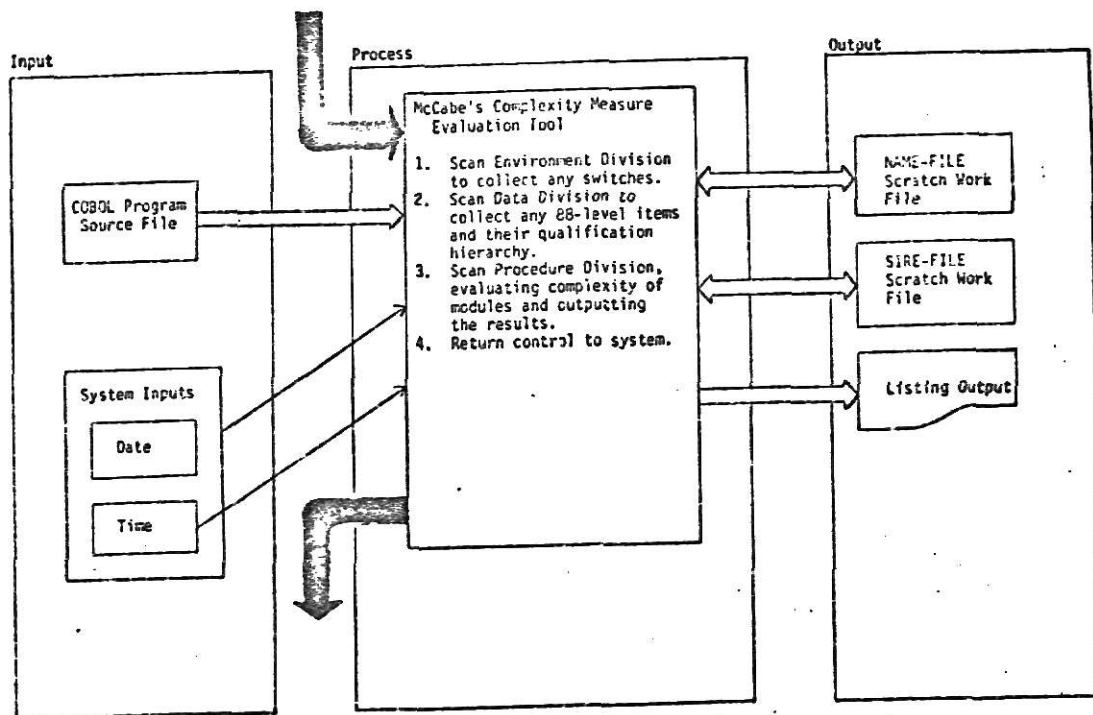


Figure C.1 General Program Flow Through McCabe's Complexity Measure Evaluation Tool.

4.0 Component Description

4.1 Local Data

Local Data for the McCabe's Complexity Measure Evaluation Tool consists of the following:

<u>NAME</u>	<u>DESCRIPTION</u>
SOURCE-LINE	The current source line of COBOL text being scanned.
CONDITION-NAME-ENTRY	The structure containing a condition-name, its qualification hierarchy, and other associated information, which is retained in the SIRE-FILE.
SOURCE-WORD	The current source word of the COBOL text being scanned.
SAVE-WORD	Used to hold a copy of SOURCE-WORD.
QUALIFICATION-ENTRIES	Holds the qualification hierarchy of an item being scanned in the Procedure Division, for comparison with SIRE-FILE entries.
THE-TIME	The time of day, in hours, minutes, and seconds. Retrieved from the system.
HEADER	Information printed at top of each page in the listing output.
PRINT-LINE-1 through PRINT-LINE-8	These contain the formats of information to be placed in the listing output.
THE-DATE	The date, as retrieved from the system.
ACCUMULATIVE-COMPLEXITY	The total complexity of the COBOL source input program.
GO-TO-COMPLEXITY	Used as a temporary structure to retain complexity of potential GO-TO-DEPENDING situation.

QUALIFICATION-COUNTER	Index of depth of qualification of data-name being scanned.
PAGE-COUNT	Counter incremented for each page of the listing output.
NUMBER-OF-CONDITION-NAMES	Counter incremented for each new condition-name encountered.
SIRE-KEY	Relative key for the SIRE-FILE.
NAME-KEY	Relative key for the NAME-FILE.
PTR	Index into source line being scanned.
START-OF-WORD	Index indicating where the starting position in the source line of the current word being scanned is.
END-OF-WORD	Index indicating position of end character in the source line of the current word being scanned.
WORD-SIZE	Size in characters of the current word being scanned.
SOURCE-PTR	Index into the source line used in scanning literals.
PTR-TEMP	Temporary variable to hold indices into the current source line being scanned.
DELIMITER-TEMP	Temporary value used to hold source word delimiters in the scanning of subscripts.
SOURCE-WORD-TEMP	Temporary storage for retaining source word values.
SOURCE-STOP	Contains the delimiter of the current source word being scanned.
SIRE-TABLE-SUBSCRIPT	Index into SIRE-TABLE of CONDITION-NAME-ENTRY.
SUBSCRIPT-TEMP	Index used for indicating position of the end of the source word being scanned when skipping past parentheses in the Procedure Division.
CURRENT-LEVEL	Present level-number of data-item being scanned.

LAST-LEVEL	Level-number of last data-item scanned.
RUNNING-SECTION-COUNT	The running total of the number of paragraphs in the current section being scanned.
RUNNING-PROGRAM-COUNT	The running total of the number of paragraphs in the source program.
RUNNING-SECTION-TOTAL	The running total of the complexity of the paragraphs of the section currently being scanned.
RUNNING-PROGRAM-TOTAL	The running total of the complexity of the paragraphs in the source program.
SECTION-AVERAGE	The average complexity of the paragraphs in the last section scanned.
PROGRAM-AVERAGE	The average complexity of the paragraphs in the source program.
SIRE-NAME	The id-name of a qualifier.
FILE-FLAG	A flag used to indicate whether the source program being scanned has fixed or variable length records.
EOF-FLAG	A flag used to indicate that the scanning of the source program has reached the end of the source file.
AREA-A-FLAG	A flag used to indicate whether the AREA-A of the current source line is blank or non-blank.
CONDITION-FIND-FLAG	A flag used to indicate that a condition name has been found in the search of the SIRE-FILE that matches the condition-name in the conditional expression being scanned.
DONE-SW	A general purpose flag.
CONSUME-SW	A flag used in the scanning of literals.
QUOTE-SW	A flag used in the scanning of literals.

CONDITIONAL-SITUATION-SW	A flag used to indicate that a conditional situation has been found in the current source line being scanned.
NO-NOT-SW	A flag used to indicate the presence of 'NOT' in the current source line.
MORE-CONDITION-SW	A flag used to indicate whether the current conditional expression being scanned is compound or not.
RELATIONAL-OPERATOR-SW	A flag used to indicate the presence of a relational operator in the current source line being scanned.
GO-TO-DEPENDING-SW	A flag used to indicate that a GO-TO-DEPENDING has been found in the current source line being scanned.
FIRST-TIME-THRU-SW	A flag used to indicate whether or not this is the first module being scanned.
LINE-NUMBER	The line-number of the current source line being scanned.
LINE-NUMBER-DISPLAY	Used for conversion of numeric LINE-NUMBER to alphanumeric value for display to the CRT device.
ADD-ONE-SW	A flag used to indicate if a line of code has been encountered in the current module being scanned, or if the module is empty.
PRINT-TOTAL-N-AVERAGE-SW	A flag used to indicate whether the situation exists where the total and average complexity should be printed.

4.2 Routines

The following routines are part of McCabe's Complexity Measure Evaluation Tool:

<u>ROUTINE</u>	<u>FUNCTION</u>
MAIN	Outermost routine (Driver).
GET-WORD	Isolate the next token in the COBOL source program.
READ-NEXT	Inputs the next character of the COBOL source file.
ISOLATE-WORD	Isolate a token in the COBOL source program.
CONSUME-LITERAL	Isolate a literal token in the COBOL text.
EXAMINE-AND-ISOLATE-SUBSCRIPTS	Strip off and examine subscripts.
GET-TO-TOP-OF-PAGE	Slew the output listing to the top of the page and print the heading with page number.
GET-TIME-AND-DATE	Accept time and date for listing from the system.
PROCESS-PROGRAM-ID-PARAGRAPH	Scan for program-id to put in the listing.
COLLECT-CONDITION-NAMES	Collect condition-name entries in the Environment and Data Divisions.
COLLECT-ANY-SWITCHES	Scan the COBOL source program, looking for any switches.
COLLECT-ANY-88-LEVELS	Scan the COBOL source program, collecting all the 88 level items found in the Data Division.
ANALYZE-CURRENT-LEVEL	Examine the level number of the item being scanned. If the level number indicates that the item is above the last item scanned in the hierarchy of the data structure being scanned, remove the unrelated sires from the

	SIRE-TABLE. Then place the level of the item being scanned and the item in the SIRE-TABLE.
REGISTER-88-LEVEL	Scan an 88 level item and register it and its corresponding complexity.
OUTPUT-CONDITION-ENTRY	Output an entry to the SIRE-FILE.
OUTPUT-SIRE-NAME	Output an entry to NAME-FILE.
PROCESS-PROCEDURE-DIVISION	Analyze the complexity of the paragraphs found in the Procedure Division.
GET-MODULE-COMPLEXITY	Analyze the complexity of an individual paragraph.
FIND-CONDITIONAL-SITUATIONS	Scan the COBOL source program, looking for a conditional expression.
SCAN-FOR-GOTO-DEPENDING	For every possible GOTO DEPENDING situation, accumulate the potential complexity. If the DEPENDING phrase is found, add to complexity totals the accumulated complexity for the GOTO DEPENDING.
CHECK-MODULE-START	Scan the COBOL source program, looking for the start of a section or paragraph.
ANALYZE-COMPLEXITY	Scan a conditional, calculating its complexity.
GET-PAST-AND-REMOVE-PARENS	Scan past parentheses.
SCAN-FOR-NOT	Scan for NOT, setting NO-NOT-SW as to whether one is found or not found.
SCAN-QUALIFICATION-N-SUBSCRIPT	Scan any qualification and associated subscripts, scanning past the subscripts, and placing the qualification items in the table called QUALIFICATION-ENTRIES.
SCAN-FOR-IS-N-NOT	Scan past IS and NOT if they are found.
SCAN-FOR-OR-N-AND	Scan for the connectors 'OR' and 'AND', which indicate compound conditionals.

CHECK-IF-RELATIONAL-OPERATOR

If the current source word is a relational operator, then the conditional being scanned is a relational condition, and each simple relational condition, whether or not it is a member of a complex conditional expression, has a complexity value of one.

CHECK-OUT-CONDITION

Determine the complexity of the conditional being scanned.

GO-THRU-CONDITION-NAMES

Look through the SIRE-FILE for an entry that matches the condition-name currently pointed to in the source program.

PRINT-ROUTINES

Output the results to the PRINT-FILE.

5.0 Development and Maintenance

5.1 Development Systems

The primary development system used for McCabe's Complexity Measure Evaluation Tool was an I8270 with 1024k bytes of memory, 6560 and 6530 disk drives, a 600-lpm band printer, and the IMOS V operating system. Both compilation and execution may be performed on this system.

5.2 Maintenance Considerations

In order to facilitate maintainability, the COBOL source program for McCabe's Complexity Measure Evaluation Tool, when compiled using the Debug option of the system execute statement, and executed, will display additional information to the CRT device, including the source line and the line number of the source line being scanned.

Of assistance in maintenance is the I-Family COBOL interactive Debug facility, which is activated when using the Debug option on the system execute statement when executing the McCabe's Complexity Measure Evaluation Tool.

APPENDIX D

I-Spec For Halstead's Tool

The following is the Implementation Specification for the tool to evaluate Halstead's Software Science Metrics.

IMPLEMENTATION SPECIFICATION
FOR
HALSTEAD'S METRICS EVALUATION TOOL

1.0 Overview

1.1 Scope

This document specifies the implementation details of the Halstead's Metrics Evaluation Tool.

1.2 General Description

The Halstead's Metrics Evaluation Tool, which operates on systems which support NCR I-Family COBOL, is designed to scan a COBOL source program in terms of the Software Science metrics defined by M. Halstead, and output the results to a listing file. The files to be used by the Halstead's Metrics Evaluation Tool must be established while under control of the operating system.

1.3 References

The following references should be consulted for additional information on the concepts discussed in this specification:

- (1) Meals, Randall R., Functional Specification for Halstead's Metrics Evaluation Tool. (See Appendix B.)
- (2) Halstead, M. H., "Advances in Software Science," Advances in Computers, Vol. 18, Academic Press, Inc., New York, 1979.

- (3) Lyusternik, L. A., O. A. Chervonenkis, and A.R. Yanpol'skii, Handbook for Computing Elementary Functions, Pergamon Press Ltd., Oxford, 1965.

2.0 Environment

The Halstead's Metric Evaluation Tool is designed for use on any of the members of NCR's I-Family of computing systems. Currently, the hardware systems forming the I-Family include 8080-based, 605-based, and 8400-based user systems. The operating systems forming the I-Family environment include the IDPS, MDPS, IMOS III, IMOS V, and IRX operating systems. Execution of I-Family COBOL is also possible under the MCITS and TCOS systems.

3.0 Design Summary

The Halstead's Metrics Evaluation Tool is designed as a utility for COBOL application developers. It provides a means of automating the calculation of the metrics of Halstead's Software Science discipline. The tool consists of a COBOL program, which accepts COBOL source as input, utilizes two scratch work files, and produces a listing output of the results of the measures applied to the source input. The following metrics from Software Science are utilized in the evaluation of the source input:

- 1) The number of unique operators
- 2) The number of unique operands
- 3) The total frequency of operators
- 4) The total frequency of operands
- 5) Program Vocabulary
- 6) Program Length
- 7) Estimated Program Length
- 8) Program Volume
- 9) Implementation Level
- 10) Potential Volume
- 11) Effort Metric
- 12) Programming Time
- 13) Language Level

For further explanation of the above metrics, see reference (2) in Section 1.3 of this document.

3.1 System Flow

The Halstead's Metrics Evaluation Tool is invoked as a normally executable COBOL program via the appropriate system command, with its input in logical file I and its output in logical file PRINT-FILE. There must also be assigned scratch work files associated with the logical files OPERAND and LINK.

3.2 Logic Flow

The Halstead's Metrics Evaluation Tool is a COBOL program. It accepts as input COBOL source. The tool scans the COBOL source, and first locates the beginning of the Procedure Division. The tool then scans the Procedure Division of the source input, examining each token and determining if it is an operator or an operand. This is accomplished by comparing the token against a table containing the known operators of the COBOL language. If the token is found in the table of operators, then the count of occurrences associated with that operator is incremented by one, and the running count of the frequency of operators in the program is also incremented by one. If the token is not found in the table of operators, it is assumed to be an operand. In this case, a search is made to see if a reference to the current token has already been made. One of the work files contains a set of links chaining the contents of the second work file, which contains previously scanned operands and their individual frequencies of occurrence. A chain exists for each letter of the alphabet, and a chain exists for all non-alphabetic characters. If the token is not matched by one of the operands in the work file, an entry is created so that the token is added to the set of known operands. The count of occurrences associated with the

entry in the work files for that operand is incremented by one, and the running count of the frequency of operands in the program is also incremented by one. The scanning continues until the end of the program. At that point, the tool calculates the various metrics and outputs to the listing file a page containing the measured and computed metrics. This is followed by separate lists of the operators and operands and their frequencies, output in order of frequency.

3.3 Program Flow

The general program flow through the Halstead's Metrics Evaluation Tool is shown in Figure D.1.

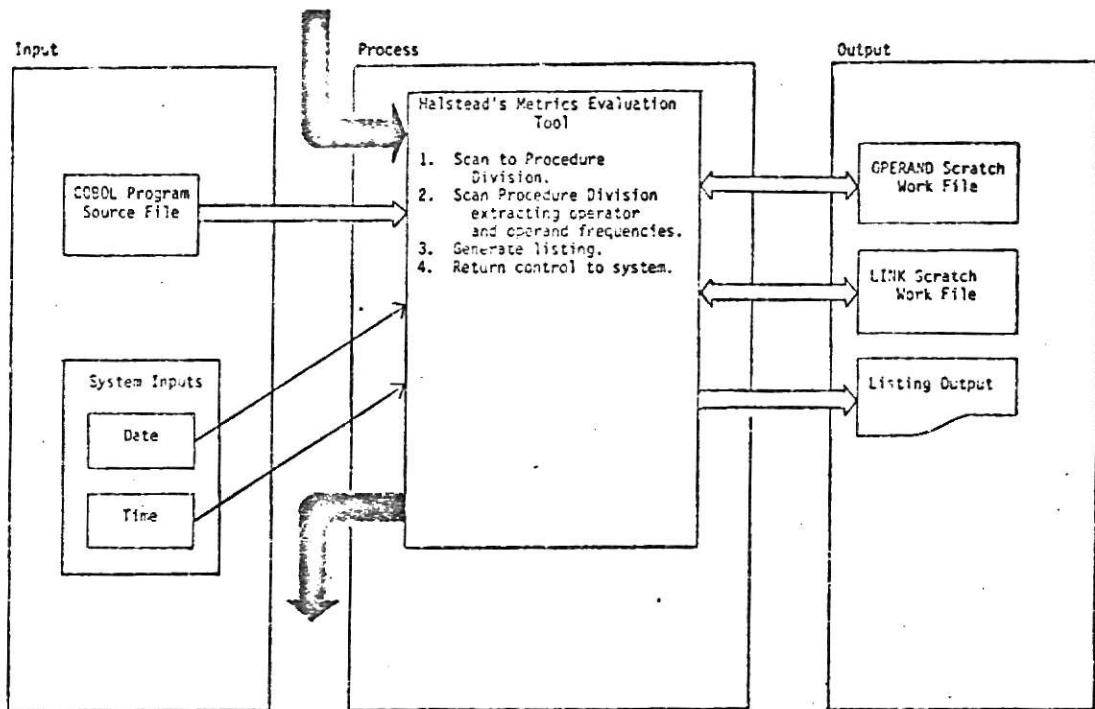


Figure D.1 General Program Flow Through Halstead's Metrics Evaluation Tool.

4.0 Component Description

4.1 Local Data

Local Data for the Halstead's Metrics Evaluation Tool consists of the following:

<u>NAME</u>	<u>DESCRIPTION</u>
SOURCE-LINE	The current source line of COBOL text being scanned.
SOURCE-WORD	The current source word of the COBOL text being scanned.
SAVE-WORD	Used to hold a copy of SOURCE-WORD.
OPERATOR-ENTRIES	Table which holds the id-name of known COBOL operators and their frequencies.
ALPHABET-ENTRIES	Holds the links chaining together operand id-names using the first character of the operand id-names as an index..
OPERAND-HEAD-LINKS	The head links of the chain of link elements for operands and their frequencies.
OPERAND-ENTRY	Contains an operand id-name and its frequency.
THE-TIME	The time of day, in hours, minutes, and seconds. Retrieved from the system.
HEADER	Information printed at top of each page in listing output.
PRINT-LINE-1 through PRINT-LINE-20	These contain the formats of information to be placed in the listing output.
THE-DATE	The date, as retrieved from the system.
PAGE-COUNT	Counter incremented for each page of the listing output.

PTR	Index into the source line being scanned.
START-OF-WORD	Index indicating where the starting position in source line of the current word being scanned is.
END-OF-WORD	Index indicating position of end character in the source line of the current work being scanned.
WORD-SIZE	Size in characters of the current word being scanned.
SOURCE-PTR	Index into the source line used in scanning literals.
PTR-TEMP	Temporary variable to hold indices into the current source line being scanned.
SUBSCRIPT-TEMP	Index used for indicating position of the end of the source word being scanned when skipping past parentheses in the Procedure Division.
DELIMITER-TEMP	Temporary value used to hold source word delimiters in the scanning of subscripts.
SOURCE-WORD-TEMP	Temporary storage for retaining source word values.
SOURCE-STOP	Contains the delimiter of the current source word being scanned.
SAVE-STOP	Used to hold a copy of SOURCE-STOP.
LINE-NUMBER	The line-number of the current source line being scanned.
LINE-NUMBER-DISPLAY	Used for conversion of numeric LINE-NUMBER to alphanumeric value for display to the CRT device.
OPERAND-KEY	Relative key for the OPERAND scratch work file.
LINK-KEY	Relative key for the LINK scratch work file.

NUMBER-OF-LINKS	Contains total number of items in the linked lists of operands.
LAST-LINK	Contains value of the last item to be examined in a link list of operands.
LINK-INDEX	Contains value corresponding to which link list head currently corresponds to the first character of the current operand being scanned.
ITEM-BEING-CONSIDERED	Used to retain value corresponding to the current item examined in the outputting of operators and operands by frequency.
ONE-TO-PRINT	The value corresponding to the index of the operator or operand to be output.
LAST-ONE-PRINTED	The value corresponding to the index of the last operator or operand which was output with its frequency.
CURRENT-HIGH	The value of the highest frequency to be encountered during a pass through outputting the operators or operands, that is less than or equal to the frequency of the last item output.
LAST-HIGH	The frequency of the last operator or operand output.
NUMBER-OF-ITEMS-PRINTED	The total number of operators or operands output.
NUMBER-OF-UNIQUE-OPERANDS	The total number of unique operands encountered in the source input.
NUMBER-OF-UNIQUE-OPERATORS	The total number of unique operators encountered in the source input.
TOTAL-FREQUENCY-OF-OPERANDS	The total number of operands encountered in the source input.
TOTAL-FREQUENCY-OF-OPERATORS	The total number of operators encountered in the source input.
PROGRAM-VOCABULARY	The computed value of the Program Vocabulary metric.

PROGRAM-LENGTH	The computed value of the Program Length metric.
PROGRAM-VOLUME	The computed value of the Program Volume metric.
IMPLEMENTATION-LEVEL	The computed value of the Implementation metric.
POTENTIAL-VOLUME	The computed value of the Potential Volume metric.
EFFORT-METRIC	The computed value of the Effort metric.
ESTIMATE-OF-PROGRAM-LENGTH	The computed value of the Estimate of Program Length metric.
PROGRAMMING-TIME-SECONDS	The computed value of the Programming Time metric in seconds.
PROGRAMMING-TIME-MINUTES	The computed value of the Programming Time metric in minutes.
PROGRAMMING-TIME-HOURS	The computed value of the Programming Time metric in hours.
LANGUAGE-LEVEL	The computed value of the Language Level metric.
PASSED-TO-LOG-ROUTINE	The value of which the base two logarithm is to be calculated.
LOG-BASE-TWO-RESULT	A calculated base two logarithm corresponding to the PASSED-TO-LOG-ROUTINE value.
TEMP-1 through TEMP-7	Temporary variables used for producing the computed metrics.
HALF-LOG	Used in the calculation of base two logarithms.
LOG-ADDEND	Used in the calculation of base two logarithms.
LOG-E	Used to hold the natural logarithm corresponding to the PASSED-TO-LOG-ROUTINE value, and used as an intermediate step in arriving at the equivalent base two logarithm.

MULTIPLIER	Used in the calculation of base two logarithms.
FRACTION	Used in the calculation of base two logarithms.
I	Index used in the calculation of base two logarithms.
X-PLUS-1	Used in the calculation of base two logarithms.
X-MINUS-1	Used in the calculation of base two logarithms.
CONVERSION-VALUE	The value by which a natural logarithm is multiplied to arrive at an equivalent base two logarithm.
FILE-FLAG	A flag used to indicate whether the source program being scanned has fixed or variable length records.
EOF-FLAG	A flag used to indicate that the scanning of the source program has reached the end of the source file.
AREA-A-FLAG	A flag used to indicate whether the AREA-A of the current source line is blank or non-blank.
CONSUME-SW	A flag used in the scanning of literals.
QUOTE-SW	A flag used in the scanning of literals.
DONE-SW	A general purpose flag.
OPERAND-FIND-FLAG	A flag used to indicate whether the current operand being scanned has been previously encountered and thus been placed in the appropriate link list.

4.2 Routines

The following routines are part of the Halstead's Metrics Evaluation Tool:

<u>ROUTINE</u>	<u>FUNCTION</u>
MAIN	Outermost routine (Driver).
GET-WORD	Isolate the next token in the COBOL source program.
READ-NEXT	Inputs the next character of the COBOL source file.
ISOLATE-WORD	Isolate a token in the COBOL source program.
CONSUME-LITERAL	Isolate a literal token in the COBOL text.
EXAMINE-AND-ISOLATE-SUBSCRIPTS	Strip off and examine subscripts.
GET-TO-TOP-OF-PAGE	Slew the output listing to the top of the page and print the heading with page number.
PRINT-BLANK-LINE	Output a blank line to the output listing.
GET-TIME-AND-DATE	Accept time and date for listing from the system.
INITIALIZE-OPERATOR-TABLE	Initialize the entries in the OPERATOR-TABLE.
INITIALIZE-ALPHABET-TABLE	Initialize the entries in the ALPHABET-TABLE and in the HEAD-LINK-LIST
PROCESS-PROGRAM-ID-PARAGRAPH	Scan for program-id to put in the listing.
PROCESS-PROCEDURE-DIVISION	Scan the source program's Procedure Division and collect all occurrences of operators and operands.
GET-PAST-AND-REMOVE-PARENS	Scan for any parentheses and remove .. them from the source word to be examined.

REGISTER-OPERATOR	Enter the operator currently being scanned into the operator table by incrementing the counter of the entry associated with the passed index.
REGISTER-OPERAND	Register the current operand being scanned by placing it in the OPERAND-FILE if it does not already have an associated entry, or if it does, by incrementing the counter of that entry. In order to expedite the search for a given operand a linked list is maintained for each letter of the alphabet, plus a linked list of all non-alphabetic characters.
COMPUTE-MEASURES	Compute the metrics defined by Halstead and his co-workers from the information collected.
CALCULATE-PROGRAM-VOCABULARY	Compute the Program Vocabulary metric.
CALCULATE-PROGRAM-LENGTH	Compute the Program Length metric.
CALCULATE-ESTIMATE-OF-LENGTH	Compute the Estimate of Program Length metric.
CALCULATE-PROGRAM-VOLUME	Compute the Program Volume metric.
COMPUTE-LOG-BASE-TWO	Calculate the base two logarithm of the passed value.
CALCULATE-IMPLEMENTATION-LEVEL	Compute the Implementation Level metric.
CALCULATE-POTENTIAL-VOLUME	Compute the Potential Volume metric.
CALCULATE-EFFORT-METRIC	Compute the Effort metric.
CALCULATE-PROGRAMMING-TIME	Compute the Programming Time metric in terms of seconds, minutes, and hours, using Stroud's Number of 18 Elementary Mental Discriminations per second.
CALCULATE-LANGUAGE-LEVEL	Compute the Language Level metric.
OUTPUT-RESULTS-ROUTINE	Output the measured and computed metrics to the PRINT-FILE.
OUTPUT-MEASURED-RESULTS	Output the results of the measured metrics to the PRINT-FILE.

OUTPUT-COMPUTED-RESULTS

Output the results of the computed metrics to the PRINT-FILE.

OUTPUT-OPERATOR-FREQUENCIES

Output the operators found in the source program in order of descending frequency to the PRINT-FILE.

OUTPUT-OPERAND-FREQUENCIES

Output the operands in the source program in order of descending frequency to the PRINT-FILE.

5.0 Development and Maintenance

5.1 Development Systems

The primary development system used for Halstead's Metrics Evaluation Tool was an I8270 with 1024K bytes of memory, 6560 and 6530 disk drives, a 600-lpm band printer, and the IMOS V operating system. Both compilation and execution may be performed on this system.

5.2 Maintenance Considerations

In order to facilitate maintainability, the COBOL source program for Halstead's Metrics Evaluation Tool, when compiled using the Debug option of the system execute statement, and executed, will display additional information to the CRT device, including the source line and the line number of the source line being scanned.

Of assistance in maintenance is the I-Family COBOL interactive Debug facility, which is activated when using the Debug option on the system execute statement when executing the Halstead's Metrics Evaluation tool.

APPENDIX E

McCabe's Tool Source Code

The following listing is the COBOL source code for the tool to evaluate McCabe's Cyclomatic Software Complexity Measure.

```

LINE  DEBUC  SEQ/NO  A...H..... ID.....
1*  IDENTIFICATION DIVISION.
2*  PROGRAM-ID.  MCCABES-MEASURE.
3*  AUTHOR.  KADALL K. MEALS.
4*  DATE-WRITTEN.  80/04/29.
5*  DATE-COMPILED.  81/03/14.
6*  ENVIRONMENT DIVISION.
7*  CONFIGURATION SECTION.
8*  SOURCE-COMPUTER.  NCR-IMOS.
9*  OBJECT-COMPUTER.  NCR-IMOS.
10*
11*
12*
13*
14*
15*
16*
17*
18*

```

LINE DEBUG SEQ/NO A..... ID.....

```

190 / INPUT-OUTPUT SECTION.
200 FILE-CONTROL.
210 SELECT FIXED-INPUT ASSIGN TO DISC.
220 SELECT VARIABLE-INPUT ASSIGN TO DISC.
230 SELECT PRINT-FILE ASSIGN TO PRINTER.
240 SELECT SIRE-FILE ASSIGN TO DISC
250 ORGANIZATION IS RELATIVE;
260 ACCESS MODE IS RANDOM;
270 RELATIVE KEY IS SIRE-KEY.
280 SELECT NAME-FILE ASSIGN TO DISC
290 ORGANIZATION IS RELATIVE;
300 ACCESS MODE IS RANDOM;
310 RELATIVE KEY IS NAME-KEY.
320
330 I-O-CONTROL.
340 SAME RECORD AREA FOR FIXED-INPUT, VARIABLE-INPUT.
350 DATA DIVISION.
360 FILE SECTION.
370 FD FIXED-INPUT
380 BLOCK CONTAINS 6 RECORDS
390 RECORD CONTAINS 80 CHARACTERS
400 LABEL RECORDS ARE STANDARD
410 VALUE OF FILE-ID IS "I".
420 01 INPUT-LINE PIC X(80).
430 *
440 FD VARIABLE-INPUT
450 BLOCK CONTAINS 512 CHARACTERS
460 RECORD VARYING 1 TO 80 CHARACTERS DEPENDING ON IO-1
470 LABEL RECORDS ARE STANDARD
480 VALUE OF FILE-ID IS "I".
490 01 INPUT-LINE PIC X(400).
500 *
510 FD PRINT-FILE
520 BLOCK CONTAINS 1 RECORDS
530 RECORD CONTAINS 132 CHARACTERS
540 LABEL RECORDS ARE OMITTED
550 LINAGE IS 58 LINES
560 LINES AT TOP 3.
570 LINES AT BOTTOM 5.
580 01 PRINT-RECORD PIC X(132).
590 *
600 FD SIRE-FILE
610 LABEL RECORDS ARE STANDARD.
620 01 SIRE-RECORD.
630 05 S-COMD-NAME PIC X(30).
640 05 S-SIRE-TABLE OCCURS 50 TIMES.
650 10 S-SIRE-NAME-POINTER PIC 9(4).
660 10 S-SIRE-LEVEL PIC X(2).
670 05 S-COMPLEXITY-VALUE PIC 9(18).
680 05 S-SWITCH-FLAG PIC 9.
690 05 S-TABLE-SIZE PIC 99.
700 *
710 FD NAME-FILE
720 BLOCK CONTAINS 17 RECORDS
730 LABEL RECORDS ARE STANDARD.
740 01 NAME-RECORD PIC X(30).

```

NCR-1405 COBOL COMPILE VERSION 05.12.00 01/03/14 10:10:00 SI-MCCABES 80-DNCCABES

LINE DEBUG SEQ/NO A....B..... ID.....

```

75* / WORKING-STORAGE SECTION.
76*
77*
78*
79* 01 SOURCE-LINE.
80*    05 SEQUENCE-NUMBER          PIC X(6).
81*    05 COMMENT-FIELD           PIC X.
82*    05 COMMENT-OR-DEBUG        VALUES ARE "M", "D", "W".
83*    05 LINE-CONTINUATION       VALUE IS "-".
84*    05 TEXT-LINE               PIC X(65).
85*    05 ID-FIELD                PIC X(8).
86* 01 SOURCE-LINE-REDEFINED REDEFINES SOURCE-LINE.
87*    05 SOURCE-LINE-ITEMS OCCURS 80 TIMES.
88*    10 SOURCE-LINE-ITEM PIC X.
89*
90* 01 CONDITION-NAME-ENTRY.
91*    05 CONDITION-NAME          PIC X(30).
92*    05 SIRE-TABLE OCCURS 50 TIMES.
93*    10 SIRE-NAME-POINTER      PIC 9(4).
94*    10 SIRE-LEVEL              PIC X(2).
95*    05 COMPLEXITY-VALUE       PIC 9(10).
96*    05 SWITCH-FLAG            PIC 9.
97*    05 IS-A-SWITCH            VALUE 1.
98*    05 NOT-A-SWITCH           VALUE 0.
99*    05 TABLE-SIZE            PIC 99.
100*
101* 01 SOURCE-WORD PIC X(120).
102*    05 A-LEVEL-NUMBER VALUES ARE "77", "88", "FD".
103*    "01" THRU "49", "66".
104*    05 RELATIONAL-OPERATOR VALUES ARE "<", ">", "=",
105*    "GREATER", "LESS", "EQUAL".
106*    05 A-SWITCH VALUES ARE "SWITCH-1" THRU "SWITCH-8".
107*    05 CLASS-CONDITION VALUES ARE "POSITIVE", "NEGATIVE",
108*    "ZERO", "NUMERIC", "ALPHANUMERIC", "ALPHABETIC".
109* 01 SOURCE-WORD-REDEFINED REDEFINES SOURCE-WORD.
110*    05 SOURCE-WORDS OCCURS 120 TIMES.
111*    10 SOURCE-WORD-ITEM PIC X.
112*
113* 01 SAVE-WORD PIC X(30).
114*
115* 01 QUALIFICATION-ENTRIES.
116*    05 QUALIFICATION-TABLE OCCURS 50 TIMES.
117*    10 QUALIFICATION-ITEM PIC X(30).
118*
119* 01 THE-TIME.
120*    05 HOURS PIC 99.
121*    05 MINUTES PIC 99.
122*    05 SECONDS PIC 99.

```

LINE DENS SEQ/NO A...B..... ID.....

```

1220 / 01 HEADER.
1230 05 FILLER PIC X(8) VALUE IS "MCCABES".
1240 05 FILLER PIC X VALUE IS SPACES.
1250 05 FILLER PIC X(10) VALUE IS "COMPLEXITY".
1260 05 FILLER PIC X VALUE IS SPACES.
1270 05 FILLER PIC X(7) VALUE IS "MEASURE".
1280 05 FILLER PIC X(10) VALUE IS SPACES.
1290 05 FILLER PIC X(12) VALUE IS "PROGRAM-NAME".
1300 05 FILLER PIC X(3) VALUE IS " ".
1310 05 PROGRAM-NAME PIC X(30).
1320 05 FILLER PIC X(15) VALUE IS SPACES.
1330 05 DATE-OUT PIC 99/99/99.
1340 05 FILLER PIC X(15) VALUE IS SPACES.
1350 05 HOURS-OUT PIC 99.
1360 05 FILLER PIC X VALUE IS "H".
1370 05 MINUTES-OUT PIC 99.
1380 05 FILLER PIC X VALUE IS "M".
1390 05 SECONDS-OUT PIC 99.
1400 05 FILLER PIC X(15) VALUE IS SPACES.
1410 05 FILLER PIC X(4) VALUE IS "PAGE".
1420 05 FILLER PIC X VALUE IS SPACES.
1430 05 PAGE-NUMBER PIC X(3).
1440
1450
1460 01 PRINT-LINE-1.
1470 05 FILLER PIC X(15) VALUE IS SPACES.
1480 05 FILLER PIC X(12) VALUE IS "SECTION-NAME".
1490 05 FILLER PIC X(2) VALUE IS SPACES.
1500 05 FILLER PIC X VALUE IS " ".
1510 05 FILLER PIC X(12) VALUE IS SPACES.
1520 05 SECTION-NAME-1 PIC X(30).
1530 05 FILLER PIC X(4) VALUE IS SPACES.
1540 05 FILLER PIC X(14) VALUE IS "SECTION-NUMBER".
1550 05 FILLER PIC X VALUE IS SPACES.
1560 05 FILLER PIC X VALUE IS " ".
1570 05 FILLER PIC X VALUE IS SPACES.
1580 05 SECTION-NUMBER PIC XX.
1590
1600 01 PRINT-LINE-2.
1610 05 FILLER PIC X(5) VALUE IS SPACES.
1620 05 FILLER PIC X(12) VALUE IS "SECTION-NAME".
1630 05 FILLER PIC X(3) VALUE IS SPACES.
1640 05 FILLER PIC X(13) VALUE IS " ".
1650 05 FILLER PIC X(13) VALUE IS SPACES.
1660 05 SECTION-NAME-2 PIC X(30).
1670
1680 01 PRINT-LINE-3.
1690 05 FILLER PIC X(10) VALUE IS SPACES.
1700 05 FILLER PIC X(14) VALUE IS "PARAGRAPH-NAME".
1710 05 FILLER PIC X(3) VALUE IS SPACES.
1720 05 FILLER PIC X VALUE IS " ".
1730 05 FILLER PIC X(3) VALUE IS SPACES.
1740 05 MODULE-NAME PIC X(30).

```

NCR-1M05 COBOL COMPILATION VERSION 05.12.00 81/03/14 18118:00 51-MCCABES BO-ORCCABES

LINE DEBJG SEQ/NO A...B.....10.....

```

1750 / 01 PRINT-LINE-4.
1760 05 FILLER PIC X(15) VALUE IS SPACES.
1770 05 FILLER PIC X(10) VALUE IS "COMPLEXITY".
1780 05 FILLER PIC X(3) VALUE IS " ".
1790 05 COMPLEXITY PIC Z(17)9.
1800
1810
1820
1830 01 PRINT-LINE-5.
1840 05 FILLER PIC X(10) VALUE IS SPACES.
1850 05 FILLER PIC X(5) VALUE IS "TOTAL".
1860 05 FILLER PIC X VALUE IS SPACES.
1870 05 FILLER PIC X(10) VALUE IS "COMPLEXITY".
1880 05 FILLER PIC X VALUE IS SPACES.
1890 05 FILLER PIC X(3) VALUE IS "FORM".
1900 05 FILLER PIC X VALUE IS SPACES.
1910 05 FILLER PIC X(4) VALUE IS "THIS".
1920 05 FILLER PIC X(7) VALUE IS "SECTION".
1930 05 FILLER PIC X VALUE IS SPACES.
1940 05 FILLER PIC X VALUE IS " ".
1950 05 FILLER PIC X VALUE IS SPACES.
1960 05 RUNNING-SECTION-TOTAL-OUT PIC Z(17)9.
1970
1980
1990 01 PRINT-LINE-6.
2000 05 FILLER PIC X(10) VALUE IS SPACES.
2010 05 FILLER PIC X(7) VALUE IS "AVERAGE".
2020 05 FILLER PIC X VALUE IS SPACES.
2030 05 FILLER PIC X(10) VALUE IS "COMPLEXITY".
2040 05 FILLER PIC X VALUE IS SPACES.
2050 05 FILLER PIC X(3) VALUE IS "FORM".
2060 05 FILLER PIC X VALUE IS SPACES.
2070 05 FILLER PIC X(4) VALUE IS "THIS".
2080 05 FILLER PIC X(7) VALUE IS "SECTION".
2090 05 FILLER PIC X VALUE IS SPACES.
2100 05 FILLER PIC X VALUE IS " ".
2110 05 FILLER PIC X VALUE IS SPACES.
2120 05 SECTION-AVERAGE-OUT PIC Z(14)9(4).
2130
2140 01 PRINT-LINE-7.
2150 05 FILLER PIC X(10) VALUE IS SPACES.
2160 05 FILLER PIC X(5) VALUE IS "TOTAL".
2170 05 FILLER PIC X VALUE IS SPACES.
2180 05 FILLER PIC X(10) VALUE IS "COMPLEXITY".
2190 05 FILLER PIC X VALUE IS SPACES.
2200 05 FILLER PIC X(3) VALUE IS "FORM".
2210 05 FILLER PIC X VALUE IS SPACES.
2220 05 FILLER PIC X(4) VALUE IS "THIS".
2230 05 FILLER PIC X VALUE IS SPACES.
2240 05 FILLER PIC X(7) VALUE IS "PROGRAM".
2250 05 FILLER PIC X VALUE IS SPACES.
2260 05 FILLER PIC X VALUE IS " ".
2270 05 FILLER PIC X VALUE IS SPACES.
2280 05 RUNNING-PROGRAM-TOTAL-OUT PIC Z(17)9.

```

LINE DEBUG SEQ/NO A.....: 10.....

```

229* / 01 PRINT-LINE-8.
230* 05 FILLER VALUE IS SPACES.
231* 05 FILLER PIC X(7) VALUE IS "AVERAGE".
232* 05 FILLER PIC X(7) VALUE IS SPACES.
233* 05 FILLER PIC X(10) VALUE IS "COMPLEXITY".
234* 05 FILLER PIC X(10) VALUE IS SPACES.
235* 05 FILLER PIC X(3) VALUE IS "FOR".
236* 05 FILLER PIC X(4) VALUE IS "THIS".
237* 05 FILLER PIC X(7) VALUE IS SPACES.
238* 05 FILLER PIC X(7) VALUE IS "PROGRAM".
239* 05 FILLER PIC X(7) VALUE IS SPACES.
240* 05 FILLER PIC X(7) VALUE IS " ".
241* 05 FILLER PIC X(7) VALUE IS SPACES.
242* 05 FILLER PIC X(7) VALUE IS " ".
243* 05 FILLER PIC X(7) VALUE IS SPACES.
244* 05 PROGRAM-AVERAGE-OUT PIC Z(14).9(4).

```


MSB-INDS COBOL COMPILATION VERSION 05.12.00 81/03/14 16:18:00 ST-MCCARES RO-ONCCARES

```

LINE  DEBUC SEQ/NO  A.....R.....ID.....
2450  /
2460  77 THE-DATE                PIC 9(6).
2470  77 ACCUMULATIVE-COMPLEXITY  PIC 9(18).
2480  77 GO-TU-COMPLEXITY        PIC 9(18).
2490  77 QUALIFICATION-COUNTER    PIC 9(18).
2500  77 PAGE-COUNT              PIC 9(3).
2510  77 NUMBER-OF-CONDITION-NAMES  PIC 9(18).
2520  77 SIRE-KEY               PIC 9(18).
2530  77 NAME-KEY               PIC 9(18).
2540  77 PTR                    PIC 9(18).
2550  77 START-OF-WORD          PIC 9(18).
2560  77 END-OF-WORD            PIC 9(18).
2570  77 WORD-SIZE              PIC 9(18).
2580  77 SOURCE-PTR            PIC 9(18).
2590  77 PTR-TEMP              PIC 9(18).
2600  77 DELIMITER-TEMP        PIC XX.
2610  77 SOURCE-WORD-TEMP       PIC X(120).
2620  77 SOURCE-STOP            PIC XX.
2630  77 ID-1                  PIC 99.
2640  77 SIRE-TABLE-SUBSCRIPT   PIC 99.
2650  77 SUBSCRIPT-TEMP        PIC 9(18).
2660  77 CURRENT-LEVEL          PIC XX.
2670  77 LAST-LEVEL            PIC XX.
2680  77 RUNNING-SECTION-COUNT  PIC 9(18).
2690  77 RUNNING-PADGRAM-COUNT  PIC 9(18).
2700  77 RUNNING-SECTION-TOTAL  PIC 9(18).
2710  77 RUNNING-PADGRAM-TOTAL  PIC 9(18).
2720  77 SECTION-AVERAGE       PIC 9(14)V9(4).
2730  77 PROGRAM-AVERAGE        PIC 9(14)V9(4).

```

MCK-1M05 COBOL COMPILATION VERSION 05.12.00 01/03/14 18:10:00 S1-MCCABES 80-DMCCABES

```

LINE  DEBUG SEQ/N3 A.....: 10.....
/
2740  77 SIRE-NAME PIC X(30). VALUE IS "FILLER".
2750  88 FILLER-SIRE VALUE IS ZERO.
2760  77 FILE-F-AG PIC 9 VALUE IS ZERO.
2770  88 FIXED-LENGTH-FILE VALUE IS 1.
2780  88 VARIABLE-LENGTH-FILE VALUE IS ZERO.
2790  77 EDF-FLAG PIC 9 VALUE IS ZERO.
2800  88 NOT-END-OF-FILE VALUE IS 1.
2810  88 END-OF-FILE VALUE IS ZERO.
2820  77 AREA-A-FLAG PIC 9 VALUE IS 1.
2830  88 AREA-A-OCCUPIED VALUE IS 1.
2840  77 CONDITION-FIND-FLAG PIC 9. VALUE IS 1.
2850  88 CONDITION-NAME-FOUND VALUE IS 1.
2860  77 DONE-SW PIC 9. VALUE IS 1.
2870  88 DONE VALUE IS 1.
2880  77 CONSUME-SW PIC 9 VALUE IS ZERO.
2890  88 CONSUME-DONE VALUE IS 1.
2900  77 QUOTE-SW PIC 9 VALUE IS ZERO.
2910  88 BALANCED-QUOTE VALUE IS 1.
2920  77 CONDITIONAL-SITUATION-SW PIC 9 VALUE IS ZERO.
2930  88 A-CONDITIONAL-SITUATION VALUE IS 1.
2940  77 NO-NOT-SW PIC 9 VALUE IS ZERO.
2950  88 NO-NOTS VALUE IS 1.
2960  77 MORE-CONDITION-SW PIC 9 VALUE IS ZERO.
2970  88 MORE-CONDITION VALUE IS 1.
2980  77 RELATIONAL-OPERATOR-SW PIC 9 VALUE IS ZERO.
2990  88 NOT-A-RELATIONAL-OPERATOR VALUE IS ZERO.
3000  77 GO-TO-DEPENDING-SW PIC 9 VALUE IS 1.
3010  88 GO-TO-DEPENDING-SITUATION VALUE IS ZERO.
3020  77 FIRST-TIME-THRU-SW PIC 9 VALUE IS ZERO.
3030  88 NOT-FIRST-TIME-THRU VALUE IS 1.
3040  77 LINE-NUMBER PIC 9(4). VALUE IS ZERO.
3050  88 FIRST-TIME-THRU PIC 9(4). VALUE IS ZERO.
3060  77 LINE-NUMBER-DISPLAY PIC 9(4). VALUE IS ZERO.
3070  88 LINE-NUMBER-DISPLAY PIC 9(4). VALUE IS 1.
3080  77 ADD-ONE-SW PIC 9 VALUE IS 1.
3090  88 TIME-TO-ADD-ONE-TO-COMPLEXITY PIC 9 VALUE IS 1.
3100  77 PRINT-AVE-N-TOTAL-SW PIC 9 VALUE IS ZERO.
3110  88 PRINT-AVE-N-TOTAL VALUE IS 1.

```

```

MCR-INDS COBOL COMPILE VERSION 05.12.00  81/03/14  18113100  SI-MCCABES  BD-OMCCABES
LINE  DEBJG SEQ/NO  A...B..... ID.....
312* / PROCEDURE DIVISION.
313*
314* * DECLARATIVES.
315*
316* * OPEN-ERROR SECTION.
317*000002
318*   USE AFTER STANDARD EXCEPTION PROCEDURE ON FIXED-INPUT.
319*
320*000002
321*000002
322*000006
323*00000C
324*
325* * END DECLARATIVES.

```

LINE UEBJG SEQ/NO A..... ID.....

```

3260 /
3270 *
3280 *
3290 * LIST OF ROUTINES
3300 *
3310 *
3320 * 0 COMMON SUBROUTINES
3330 * 0.1 GET WORD
3340 * 0.1.1 READ NEXT
3350 * 0.1.2 ISOLATE WORD
3360 * 0.1.2.1 CONSUME LITERAL
3370 * 0.1.2.2 EXAMINE AND ISOLATE SUBSCRIPTS
3380 *
3390 * 0.2 GET TO TOP OF PAGE
3400 *
3410 * 1 MAIN ROUTINE
3420 *
3430 * 2 GET TIME AND DATE ROUTINE
3440 *
3450 * 3 PROCESS PROGRAM ID PARAGRAPH ROUTINE
3460 *
3470 * 4 COLLECT CONDITION NAME ROUTINE
3480 * 4.1 COLLECT ANY SWITCHES
3490 * 4.2 COLLECT ANY 88 LEVELS
3500 * 4.2.1 ANALYZE CURRENT LEVEL
3510 * 4.2.2.1 REGISTER 88 LEVEL
3520 * 4.2.2.2 OUTPUT CONDITION ENTRY
3530 * 4.2.2.2 OUTPUT SIRE NAME
3540 *
3550 * 5 PROCESS PROCEDURE DIVISION ROUTINE
3560 * 5.1 GET MODULE COMPLEXITY
3570 * 5.1.1 FIND CONDITIONAL SITUATIONS
3580 * 5.1.1.1 SCAN FOR GTD DEPENDING
3590 * 5.1.2 CHECK MODULE START
3600 * 5.1.3 ANALYZE COMPLEXITY
3610 * 5.1.3.1 GET PAST AND REMOVE PARENS
3620 * 5.1.3.2 SCAN FOR NOT
3630 * 5.1.3.3 SCAN QUALIFICATION AND SUBSCRIPT
3640 * 5.1.3.4 SCAN FOR IS AND NOT
3650 * 5.1.3.5 SCAN FOR OR AND AND
3660 * 5.1.3.6 CHECK IF RELATIONAL OPERATOR
3670 * 5.1.3.7 CHECK OUT CONDITION
3680 * 5.1.3.7.1 GO THRU CONDITION NAMES
3690 *
3700 * 6 PRINT ROUTINES
3710 *

```

NCR-1405 C380L COMPILATION VERSION 05.12.00 81/03/14 1818100 S1-MCCABES B0-DACCABES

```

LINE  DEBUG SEQ/NO  A...B.....: ID.....
372*  /
373*  *****
374*  * 1 MAIN ROUTINE
375*  *
376*  *
377*  * FUNCTION: OUTERMOST ROUTINE (DRIVER)
378*  * INPUT: NONE
379*  * OUTPUT: NONE
380*  * LOCAL DATA: FIXED-LENGTH-FILE
381*  * CALLED BY: NONE
382*  * CALLS: GET-TIME-AND-DATE
383*  *          PROCESS-PROGRAM-ID-PARAGRAPH
384*  *          COLLECT-CONDITION-NAMES
385*  *          PROCESS-PROCEDURE-DIVISION
386*  *          PRINT-SECTION-TOTAL-N-AVERAGE
387*  *          PRINT-PROGRAM-TOTAL-N-AVERAGE
388*  *
389*  *****
390*  *****
391*  * MAIN-PROGRAM SECTION.
392*  *
393*  * MAIN.
394*  * PERFORM GET-TIME-AND-DATE.
395*  * OPEN INPUT FIXED-INPUT.
396*  * OPEN OUTPUT PRINT-FILE.
397*  * PERFORM PROCESS-PROGRAM-ID-PARAGRAPH.
398*  * PERFORM COLLECT-CONDITION-NAMES.
399*  * PERFORM PROCESS-PROCEDURE-DIVISION.
400*  * PERFORM PRINT-SECTION-TOTAL-N-AVERAGE.
401*  * PERFORM PRINT-PROGRAM-TOTAL-N-AVERAGE.
402*  * IF FIXED-LENGTH-FILE
403*  *   CLOSE FIXED-INPUT
404*  *   ELSE CLOSE VARIABLE-INPUT.
405*  *   CLOSE PRINT-FILE.
406*  *   STOP RUN.

```

LINE DEBUG SEQ/NO A...B..... ID.....

```

406* /
407* *****
408* 0 COMMON SUBROUTINES
409* *****
410*
411*
412* COMMON-SUBROUTINE SECTION.
413*0000046
414* *****
415*
416* 0.1 GET WORD ROUTINE
417* *****
418*
419*
420* FUNCTION: ISOLATE THE NEXT TOKEN IN THE COBOL SOURCE
421* PROGRAM
422* PTR
423* INPUT: SOURCE-WORD
424* OUTPUT: PTR
425* LOCAL DATA: NONE
426* CALLED BY: PROCESS-PROGRAM-ID-PARAGRAPH
427* COLLECT-ANY-SWITCHES
428* COLLECT-ANY-SWITCHES
429* LOOK-FOR-88-LEVELS
430* REGISTER-88-LEVEL
431* CALCULATE-COMPLEXITY
432* ACCUMULATE-COMPLEXITY
433* FIND-CONDITIONAL-SITUATIONS
434* CHECK-MODULE-START
435* GET-PAST-AND-REMOVE-PARENS
436* SCAN-FOR-NOT
437* SCAN-QUALIFICATION-N-SUBSCRIPT
438* SCAN-QUALIFICATION
439* SCAN-FOR-IS-N-NOT
440* SCAN-FOR-OR-N-AND
441* CHECK-IF-RELATION-OPERATOR
442* HEAD-NEXT
443* ISOLATE-WORD
444* *****
445*
446*0000046
447*0000046
448*0000046
449*
450*
451*
452* GET-WORD-ACTUAL.
453*0000062
454*0000062
455*
456* IF PTR GREATER THAN 72
457* PERFORM READ-NEXT
458* PERFORM READ-NEXT
459* UNTIL NOT COMMENT-OR-DEBUG
460* OR END-OF-FILE.
461*
462* IF NOT-END-OF-FILE
463* PERFORM ISOLATE-WORD.
464*

```

NCR-1M05 COBOL COMPILATION VERSION 05.12.00 01/03/14 10:10:00 ST-MCCABES BD-MCCABES

LINE DEBUG SEQ/NO A-B-.....: ID-.....

```

461+ / .....:
462+ .....:
463+ .....:
464+ .....:
465+ .....:
466+ .....:
467+ .....:
468+ .....:
469+ .....:
470+ .....:
471+ .....:
472+ .....:
473+ .....:
474+ .....:
475+ .....:
476+ .....:
477+ .....:
478+000092 .....:
479+000092 .....:
480+ .....:
481+000098 .....:
482+ .....:
483+ .....:
484+ .....:
485+ .....:
486+000082 .....:
487+000086 .....:
488+ .....:
489+ .....:
490+000084 .....:
491+00008C .....:
492+ .....:
493+ .....:
494+ .....:
495+ .....:

```

0.1.1 READ NEXT ROUTINE
 FUNCTION: INPUTS THE NEXT CHARACTER OF THE COBOL
 SOURCE FILE
 INPUT: NONE
 OUTPUT: PTR
 SOURCE-LINE
 LINE-NUMBER
 LOCAL DATA: GET-WORD-ACTUAL
 CALLED BY: CONSUME-LITERAL-CHARACTER
 CALLS: NONE

READ-NEXT.
 MOVE SPACES TO INPUT-LINE OF VARIABLE-INPUT.
 SOURCE-LINE.
 IF FIXED-LENGTH-FILE
 READ FIXED-INPUT RECORD
 AT END MOVE 1 TO EOF-FLAG
 ELSE READ VARIABLE-INPUT RECORD
 AT END MOVE 1 TO EOF-FLAG.
 MOVE 0 TO PTR.
 MOVE INPUT-LINE OF VARIABLE-INPUT TO SOURCE-LINE.
 DISPLAY SOURCE-LINE.
 LINE 6.
 ADD 1 TO LINE-NUMBER.
 MOVE LINE-NUMBER TO LINE-NUMBER-DISPLAY.
 DISPLAY "SOURCE LINE NUMBER = ",
 LINE 20, POSITION 1,
 LINE-NUMBER-DISPLAY.
 LINE 20, POSITION 22.


```

LINE  DEBUG 560/NO A...B.....10.....
539* /
540*
541*
542* 0.1.2.1 CONSUME LITERAL ROUTINE
543*
544*
545*
546*
547*
548*
549*
550*
551*
552*
553*
554*
555*
556*
557*
558*
559*
560*
561*
562*
563*
564*
565*
566*
567*
568*
569*
570*
571*
572*
573*
574*
575*
576*
577*
578*
579*
580*
581*
582*
583*
584*
585*
586*
587*
588*
589*
590*
591*
592*
593*
594*
595*
596*
597*
598*
599*
600*
601*
602*
603*
604*
605*
606*
607*
608*
609*
610*
611*
612*
613*
614*
615*
616*
617*
618*
619*
620*
621*
622*
623*
624*
625*
626*
627*
628*
629*
630*
631*
632*
633*
634*
635*
636*
637*
638*
639*
640*
641*
642*
643*
644*
645*
646*
647*
648*
649*
650*
651*
652*
653*
654*
655*
656*
657*
658*
659*
660*
661*
662*
663*
664*
665*
666*
667*
668*
669*
670*
671*
672*
673*
674*
675*
676*
677*
678*
679*
680*
681*
682*
683*
684*
685*
686*
687*
688*
689*
690*
691*
692*
693*
694*
695*
696*
697*
698*
699*
700*
701*
702*
703*
704*
705*
706*
707*
708*
709*
710*
711*
712*
713*
714*
715*
716*
717*
718*
719*
720*
721*
722*
723*
724*
725*
726*
727*
728*
729*
730*
731*
732*
733*
734*
735*
736*
737*
738*
739*
740*
741*
742*
743*
744*
745*
746*
747*
748*
749*
750*
751*
752*
753*
754*
755*
756*
757*
758*
759*
760*
761*
762*
763*
764*
765*
766*
767*
768*
769*
770*
771*
772*
773*
774*
775*
776*
777*
778*
779*
780*
781*
782*
783*
784*
785*
786*
787*
788*
789*
790*
791*
792*
793*
794*
795*
796*
797*
798*
799*
800*
801*
802*
803*
804*
805*
806*
807*
808*
809*
810*
811*
812*
813*
814*
815*
816*
817*
818*
819*
820*
821*
822*
823*
824*
825*
826*
827*
828*
829*
830*
831*
832*
833*
834*
835*
836*
837*
838*
839*
840*
841*
842*
843*
844*
845*
846*
847*
848*
849*
850*
851*
852*
853*
854*
855*
856*
857*
858*
859*
860*
861*
862*
863*
864*
865*
866*
867*
868*
869*
870*
871*
872*
873*
874*
875*
876*
877*
878*
879*
880*
881*
882*
883*
884*
885*
886*
887*
888*
889*
890*
891*
892*
893*
894*
895*
896*
897*
898*
899*
900*
901*
902*
903*
904*
905*
906*
907*
908*
909*
910*
911*
912*
913*
914*
915*
916*
917*
918*
919*
920*
921*
922*
923*
924*
925*
926*
927*
928*
929*
930*
931*
932*
933*
934*
935*
936*
937*
938*
939*
940*
941*
942*
943*
944*
945*
946*
947*
948*
949*
950*
951*
952*
953*
954*
955*
956*
957*
958*
959*
960*
961*
962*
963*
964*
965*
966*
967*
968*
969*
970*
971*
972*
973*
974*
975*
976*
977*
978*
979*
980*
981*
982*
983*
984*
985*
986*
987*
988*
989*
990*
991*
992*
993*
994*
995*
996*
997*
998*
999*
1000*

```

MCR-1M75 CORDL COMPILATION VERSION 05.12.00 01/03/14 10:18:00 SI-MCCAMES 00-DMCCAMES

LINE DEBUG SEQ/NO A...B.....: ID.....
 595*0001DE IF BALANCED-QUOTE ;
 596* MOVE ZERO TO QUOTE-SW ;
 597* ELSE MOVE 1 TO QUOTE-SW. ;

```

LINE  DEBUG SEQ/NO A...B..... ID.....
599*  /
600*  .....
601*  .....
602*  .....
603*  .....
604*  .....
605*  .....
606*  .....
607*  .....
608*  .....
609*  .....
610*  .....
611*  .....
612*  .....
613*  .....
614*  .....
615*  .....
616*  .....
617*  .....
618*  .....
619* 0001F0  EXAMINE-AND-ISOLATE-SUBSCRIPTS.
620* 0001F0  ADD WORD-SIZE, START-OF-WORD
621*  .....
622* 0001F6  SUBTRACT 1 FROM END-OF-WORD.
623* 0001FC  IF SOURCE-LINE-ITEM (END-OF-WORD) = "]"
624*  .....
625*  .....
626*  .....
627*  .....
628*  .....
629*  .....
630*  .....
631*  .....
632*  .....
633*  .....
634*  .....
635*  .....
636*  .....
637*  .....
638* 000254  BACK-UP-TO-BEFORE-SUBSCRIPT.
639* 000256  MOVE SPACES TO SOURCE-WORD-ITEM (WORD-SIZE).
640* 00025E  SUBTRACT 1 FROM WORD-SIZE.
641* 000264  SUBTRACT 1 FROM END-OF-WORD.

```

0.1.2.2 EXAMINE AND ISOLATE SUBSCRIPTS ROUTINE
 FUNCTION: STRIP OFF AND EXAMINE SUBSCRIPTS
 INPUT: WORD-SIZE
 START-OF-WORD
 SOURCE-WORD
 SOURCE-WORD
 OUTPUT: WORD-SIZE
 START-OF-WORD
 END-OF-WORD
 SOURCE-WORD-TEMP
 DELIMITER-TEMP
 PTR-TEMP
 ISOLATE-WORD
 CALLED BY: NONE
 CALLS:

EXAMINE-AND-ISOLATE-SUBSCRIPTS.
 ADD WORD-SIZE, START-OF-WORD
 GIVING END-OF-WORD.
 SUBTRACT 1 FROM END-OF-WORD.
 IF SOURCE-LINE-ITEM (END-OF-WORD) = "]"
 AND SOURCE-WORD-ITEM (1) NOT EQUAL "("
 AND SOURCE-WORD NOT EQUAL ")"
 MOVE 1 TO PTR-TEMP
 UNSTRING SOURCE-WORD
 DELIMITED BY "(" OR SPACES
 INTO SOURCE-WORD-TEMP
 DELIMITER IN DELIMITER-TEMP
 WITH POINTER PTR-TEMP
 IF DELIMITER-TEMP = "("
 PERFORM BACK-UP-TO-BEFORE-SUBSCRIPT
 UNTIL SOURCE-LINE-ITEM (END-OF-WORD) = "("
 MOVE END-OF-WORD TO PTR
 PERFORM BACK-UP-TO-BEFORE-SUBSCRIPT.
 BACK-UP-TO-BEFORE-SUBSCRIPT.
 MOVE SPACES TO SOURCE-WORD-ITEM (WORD-SIZE).
 SUBTRACT 1 FROM WORD-SIZE.
 SUBTRACT 1 FROM END-OF-WORD.

NCR-TMDS COBOL COMPILE VERSION 05.12.00 61/01/14 18110:00 SI-MCCABES DO-DNCCABES

LINE DEBUG SEQ/NO A.....R..... ID.....

```

6420 /
6430 *
6440 *
6450 * 0.2 GET TO TOP OF PAGE
6460 *
6470 *
6480 * FUNCTION: SLEW THE OUTPUT LISTING TO THE TOP OF THE
6490 * PAGE AND PRINT THE HEADING WITH PAGE
6500 * NUMBER
6510 * INPUT: PAGE-COUNT
6520 * HEADER
6530 * OUTPUT: PAGE-COUNT
6540 * PRINT-RECORD
6550 * LOCAL DATA: PRINT-SECTION-NAME-N-NUMBER
6560 * CALLED BY: PRINT-SECTION-NAME
6570 * PRINT-MODULE-NAME
6580 * PRINT-COMPLEXITY
6590 * PRINT-SECTION-TOTAL-N-AVERAGE
6600 * PRINT-PROGRAM-TOTAL-N-AVERAGE
6610 * CALLS: NONE
6620 *
6630 *
6640 * GET-TO-TOP-OF-PAGE.
6650 * ADD 1 TO PAGE-COUNT.
6660 * MOVE PAGE-COUNT TO PAGE-NUMBER.
6670 * WRITE PRINT-RECORD FROM HEADER AFTER ADVANCING PAGE.

```

```

LINE  DEBUG SEQ/NO  A...B.....: ID.....
669*  /
670*  *
671*  * 2 GET TIME AND DATE ROUTINE
672*  *
673*  * FUNCTION: ACCEPT TIME AND DATE FOR LISTING
674*  * INPUT:  TIME AND DATE FROM SYSTEM
675*  * OUTPUT: DATE-OUT
676*  *          HOURS-OUT
677*  *          MINUTES-OUT
678*  *          SECONDS-OUT
679*  *          TIME-DATE
680*  *          LOCAL DATA: TIME-DATE
681*  *          CALLED BY: MAIN
682*  *          CALLS:  NONE
683*  *
684*  *
685*  * TIME-AND-DATE SECTION 50.
686*  *
687*  * GET-TIME-AND-DATE.
688*  * ACCEPT TIME-DATE FROM TIME.
689*  * ACCEPT TIME-DATE FROM DATE.
690*  * MOVE TIME-DATE TO DATE-OUT.
691*  * MOVE HOURS TO HOURS-OUT.
692*  * MOVE MINUTES TO MINUTES-OUT.
693*  * MOVE SECONDS TO SECONDS-OUT.

```


NCR-IMOS COBOL COMPILATION VERSION 05-12-03 01/03/14 18:18:00 SIMCCABES RU-DNCCABES

LINE DEBUG SEGNO A...B.....: 10.....

```

727* /
728*
729*
730* 4 COLLECT CONDITION NAME ROUTINE
731*
732*
733* FUNCTION: COLLECT CONDITION NAME ENTRIES IN
734* THE ENVIRONMENT AND DATA DIVISIONS
735* INPUT: SOURCE-WORD
736* OUTPUT: SIRE-FILE
737* NAME-FILE
738* LOCAL DATA: NONE
739* CALLED BY: MAIN
740* CALLS:
741* COLLECT-ANY-SWITCHES
742* COLLECT-ANY-88-LEVELS
743*
744* *****
745* CONDITION-NAME-HANDLING SECTION 52.
746*
747* COLLECT-CONDITION-NAMES.
748* OPEN OUTPUT SIRE-FILE.
749* NAME-FILE.
750* MOVE ZERO TO NAME-KEY, SIRE-KEY.
751* PERFORM GET-WORD
752* UNTIL SOURCE-WORD = "SPECIAL-NAMES"
753* OR SOURCE-WORD = "DATA"
754* OR SOURCE-WORD = "PROCEDURE".
755* IF SOURCE-WORD = "SPECIAL-NAMES"
756* PERFORM COLLECT-ANY-SWITCHES
757* UNTIL SOURCE-WORD = "DATA"
758* OR SOURCE-WORD = "PROCEDURE".
759* PERFORM COLLECT-ANY-88-LEVELS
760* UNTIL SOURCE-WORD = "PROCEDURE".
761* CLOSE SIRE-FILE, NAME-FILE.

```

```

LINE  DEBUG SEQ/NO A...H..... ID.....
762*  /
763*  .....
764*  .....
765*  * 4.1 COLLECT ANY SWITCHES .....
766*  .....
767*  .....
768*  * FUNCTION: SCAN THE COBOL SOURCE PROGRAM, LOOKING FOR .....
769*  * ANY SWITCHES .....
770*  * INPUT: SOURCE-WORD .....
771*  * OUTPUT: SOURCE-WORD .....
772*  * LOCAL DATA: NONE .....
773*  * CALLED BY: COLLECT-CONDITION-NAMES .....
774*  * CALLS: OUTPUT-CONDITION-ENTRY .....
775*  .....
776*  .....
777*  COLLECT-ANY-SWITCHES.
778*  PERFORM GET-WORD .....
779*  UNTIL A-SWITCH .....
780*  OR SOURCE-WORD = "DATA" .....
781*  OR SOURCE-WORD = "PROCEDURE". .....
782*  .....
783*  IF A-SWITCH .....
784*  PERFORM REGISTER-SWITCH .....
785*  PERFORM GET-WORD .....
786*  IF SOURCE-WORD = "ON" OR SOURCE-WORD = "OFF" .....
787*  PERFORM REGISTER-SWITCH. .....
788*  .....
789*  REGISTER-SWITCH.
790*  MOVE ZEROES TO CONDITION-NAME-ENTRY.
791*  PERFORM GET-WORD .....
792*  UNTIL SOURCE-WORD = "IS". .....
793*  PERFORM GET-WORD.
794*  MOVE SOURCE-WORD TO CONDITION-NAME.
795*  MOVE 1 TO COMPLEXITY-VALUE, SWITCH-FLAG.
796*  PERFORM OUTPUT-CONDITION-ENTRY.
    
```



```
LINE  DEBUG SEQ/NO A...B.....: ID.....
7960 /
7970 .....:
7980 .....:
7990 .....:
8000 .....:
8010 .....:
8020 .....:
8030 .....:
8040 .....:
8050 .....:
8060 .....:
8070 .....:
8080 .....:
8090 .....:
8100 .....:
8110 .....:
8120 .....:
8130 .....:
8140 .....:
8150 .....:
8160 030358 .....:
8170 030358 .....:
8180 .....:
8190 .....:
8200 .....:
8210 .....:
8220 .....:
8230 030378 .....:
8240 .....:
8250 030385 .....:
8260 .....:
8270 .....:
8280 .....:
8290 .....:
8300 .....:
8310 .....:
8320 .....:
8330 .....:
8340 030349 .....:
8350 030349 .....:
8360 03034D .....:
8370 .....:
8380 030387 .....:
8390 030389 .....:
8400 .....:
8410 .....:
8420 0303E8 .....:
8430 .....:
8440 .....:
8450 .....:
8460 .....:
8470 .....:
8480 .....:
8490 .....:
8500 .....:
8510 .....:

4.2 COLLECT ANY 88 LEVELS

FUNCTION:  SCAN THE COBOL SOURCE PROGRAM, COLLECTING
            ALL THE 88 LEVEL ITEMS FOUND IN THE
            DATA DIVISION
INPUT:     SOURCE-WORD
           SOURCE-STOP
OUTPUT:    SOURCE-WORD
           DUNE-SW
LOCAL DATA:
CALLED BY: COLLECT-CONDITION-NAMES
           GET-WORD
           REGISTER-CURRENT-LEVEL
           REGISTER-88-LEVEL
           ANALYZE-CURRENT-LEVEL

COLLECT-ANY-88-LEVELS.
PERFORM GET-WORD
UNTIL AREA-A-OCCUPIED AND
  (SOURCE-WORD = "FD"
   OR SOURCE-WORD = "01"
   OR SOURCE-WORD = "77"
   OR SOURCE-WORD = "PROCEDURE").
IF SOURCE-WORD = "FD"
  MOVE "00" TO SOURCE-WORD.
IF SOURCE-WORD NOT EQUAL "PROCEDURE"
  MOVE ZERO TO SIRE-TABLE-SUBSCRIPT
  MOVE SOURCE-WORD TO CURRENT-LEVEL
  MOVE ZEROS TO CONDITION-NAME-ENTRY
  PERFORM REGISTER-CURRENT-LEVEL
  MOVE ZERO TO DUNE-SW
  PERFORM LOOK-FOR-88-LEVELS
  UNTIL DONE.

LOOK-FOR-88-LEVELS.
MOVE CURRENT-LEVEL TO LAST-LEVEL.
PERFORM GET-WORD
UNTIL SOURCE-STOP = ". ".
PERFORM GET-WORD.
PERFORM GET-WORD
UNTIL A-LEVEL-NUMBER
  OR SOURCE-WORD = "PROCEDURE".
IF SOURCE-WORD EQUAL "PROCEDURE"
  MOVE "1" TO DUNE-SW
ELSE IF SOURCE-WORD = "88"
  PERFORM REGISTER-88-LEVEL
  ELSE IF SOURCE-WORD NOT EQUAL "FD"
    AND SOURCE-WORD NOT EQUAL "77"
    AND SOURCE-WORD NOT EQUAL "66"
    AND NOT (SOURCE-WORD = "01" AND
             LAST-LEVEL NOT EQUAL "00" )
    PERFORM ANALYZE-CURRENT-LEVEL
```

NCR-IMOS CUBOL COMPILATION VLSION 05.12.00 81/03/14 10:20:00 SI-MCCABES BU-MCCABES
 LINE DEWUC SEQ/NO A...B.....: ID.....
 8520 THRU REGISTER-CURRENT-LEVEL :
 8530 ELSE MOVE "1" TO DONE-SM. :

MCR-TMUS COBOL COMPILE VERSION 05.12.00 01/03/14 18:18:00 SI-MCCABES BD-DMCCABES

LINE DEBUG SEQ/NO A.....8.....: ID.....

```

354+ /
355+
356+
357+
358+
359+
360+
361+
362+
363+
364+
365+
366+
367+
368+
369+
370+
371+
372+
373+
374+
375+
376+
377+
378+
379+
380+
381+
382+
383+
384+
385+
386+
387+
388+
389+
390+
391+
392+
393+
394+
395+
396+
397+
398+
399+
400+
401+
402+
403+
404+
405+
406+
407+
408+
409+

```

4.2.1 ANALYZE-CURRENT-LEVEL
 FUNCTION: EXAMINE THE LEVEL NUMBER OF THE ITEM BEING
 SCANNED. IF THE LEVEL NUMBER INDICATES
 THAT THE ITEM IS ABOVE THE LAST ITEM
 SCANNED IN THE HIERARCHY OF THE DATA
 STRUCTURE BEING SCANNED, REMOVE THE
 UNRELATED SIRE FROM THE SIRE-TABLE.
 THEN PLACE THE LEVEL OF THE ITEM BEING
 SCANNED AND THE ITEM IN THE SIRE-TABLE.
 INPUT: SOURCE-WORD
 LAST-LEVEL
 SIRE-TABLE
 SIRE-TABLE-SUBSCRIPT
 NAME-KEY
 SIRE-TABLE
 SIRE-TABLE-SUBSCRIPT
 NAME-KEY
 CURRENT-LEVEL
 SIRE-NAME
 LOCAL DATA: NONE
 CALLED BY: LOOK-FOR-88-LEVELS
 CALLS: OUTPUT-SIRE-NAME
 ANALYZE-CURRENT-LEVEL.
 IF SOURCE-WORD NOT GREATER THAN LAST-LEVEL
 PERFORM BACK-UP-SIRE-TABLE.
 REGISTER-CURRENT-LEVEL.
 MOVE SOURCE-WORD TO CURRENT-LEVEL.
 ADD 1 TO SIRE-TABLE-SUBSCRIPT.
 ADD 1 TO NAME-KEY.
 MOVE CURRENT-LEVEL TO SIRE-LEVEL (SIRE-TABLE-SUBSCRIPT).
 PERFORM GET-WORD.
 MOVE SOURCE-WORD TO SIRE-NAME.
 MOVE NAME-KEY TO SIRE-NAME-POINTER (SIRE-TABLE-SUBSCRIPT).
 PERFORM OUTPUT-SIRE-NAME.
 BACK-UP-SIRE-TABLE.
 PERFORM DECREMENT-SUBSCRIPT
 UNTIL SIRE-LEVEL (SIRE-TABLE-SUBSCRIPT)
 LESS THAN SOURCE-WORD.
 PERFORM CLEAR-SIRE-TABLE-ENTRY
 VARYING SUBSCRIPT-TEMP
 FROM 50
 BY -1
 UNTIL SUBSCRIPT-TEMP EQUAL SIRE-TABLE-SUBSCRIPT.
 DECREMENT-SUBSCRIPT.
 SUBTRACT 1 FROM SIRE-TABLE-SUBSCRIPT.

MCR-IMDS COROL COMPILATION VERSION 05-12-00 81/03/14 18118100 SI-MCCARES BD-OMCCARES
 LINE DEJUG SEQ/NO A...8..... ID.....
 910*030484 CLEAR-SIRE-TABLE-ENTRY. ;
 911*030484 MOVE ZEROES TO SIRE-TABLE (SUBSCRIPT-TEMP). ;

LINE UEBJG SEQNO A...B..... ID.....

```

9120 /
9130
9140
9150
9160
9170
9180
9190
9200
9210
9220
9230
9240
9250
9260
9270
9280
9290
9300
9310
9320
9330
9340
9350
9360
9370
9380
9390
9400
9410
9420
9430
9440
9450
9460
9470
9480
9490
9500
9510
9520
9530
9540
9550
9560
9570

4.2.2 REGISTER BR LEVEL

FUNCTION:
INPUT:
OUTPUT:
CONDITION-NAME
SWITCH-FLAG
TABLE-SIZE
COMPLEXITY-VALUE
LOCAL DATA:
CALLED BY:
CALLS:

SCAN AN BR LEVEL ITCM AND REGISTER IT AND
ITS CORRESPONDING COMPLEXITY
SOURCE-WORD
SIRE-TABLE-SUBSCRIPT
CONDITION-NAME
SWITCH-FLAG
TABLE-SIZE
COMPLEXITY-VALUE
NONI
COLLECT-ANY-BR-LEVELS
GET-WORD
OUTPUT-CONDITION-ENTRY

REGISTER-BR-LEVEL.
PERFORM GET-WORD.
MOVE SOURCE-WORD TO CONDITION-NAME.
MOVE ZERO TO SWITCH-FLAG.
MOVE SIRE-TABLE-SUBSCRIPT TO TABLE-SIZE.
PERFORM CALCULATE-CONDITION-COMPLEXITY.
PERFORM OUTPUT-CONDITION-ENTRY.

CALCULATE-CONDITION-COMPLEXITY.
PERFORM GET-WORD
UNTIL SOURCE-WORD = "VALUE"
OR SOURCE-WORD = "VALUES".
PERFORM GET-WORD.
PERFORM GET-WORD.
UNTIL SOURCE-WORD NOT EQUAL "AS"
AND SOURCE-WORD NOT EQUAL "ARE".
MOVE 1 TO COMPLEXITY-VALUE.
PERFORM ACCUMULATE-COMPLEXITY
UNTIL SOURCE-STOP = "N".

ACCUMULATE-COMPLEXITY.
PERFORM GET-WORD.
IF SOURCE-WORD NOT EQUAL "THRU"
AND SOURCE-WORD NOT EQUAL "THROUGH"
AND SOURCE-WORD NOT EQUAL SPACES
ADD 1 TO COMPLEXITY-VALUE.

```


MCX-1M05 COBOL COMPILATION VERSION 05.12.00 01/03/14 10110100 SI-MCCABES DO-UMCCABES

```

LINE  DEBUG SEQ/NO  A..... ID.....
9800  /
9810  .....
9820  .....
9830  .....
9840  .....
9850  .....
9860  .....
9870  .....
9880  .....
9890  .....
9900  .....
9910  .....
9920  .....
9930  .....
9940  .....
9950  .....
9960  .....
9970  .....

          4.2.2.2  OUTPUT SIRE NAME

          FUNCTION:  OUTPUT AN ENTRY TO NAME-FILE
          INPUT:     SIRE-NAME
          OUTPUT:    NONE
          LOCAL DATA:  NONE
          CALLED BY:  REGISTER-CURRENT-LEVEL
          CALLS:     NONE

          OUTPUT-SIRE-NAME.
          WRITE NAME-RECORD FROM SIRE-NAME.
          INVALID KEY DISPLAY "RAM OUT OF NAME-FILE SPACE"
          STOP RUN.

```



```

LINE  DEBUG SEQ/ND A...B..... ID.....
10280  /
10290  .....
10300  * 5.1 GET MODULE COMPLEXITY .....
10310  * .....
10320  * .....
10330  * .....
10340  * .....
10350  * .....
10360  * .....
10370  * .....
10380  * .....
10390  * .....
10400  * .....
10410  * .....
10420  * .....
10430  * .....
10440  * .....
10450  * .....
10460  * .....
10470  * .....
10480  * .....
10490  * .....
10500  * .....
10510  * .....
10520  * .....
10530  * .....
10540  * .....
10550  * .....
10560  * .....

FUNCTIONS: ANALYZE THE COMPLEXITY OF AN INDIVIDUAL
INPUT:     PARAGRAPH
           SIRE-FILE
           NAME-FILE
           NONE
OUTPUT:    CONDITIONAL-SITUATION-SM
LOCAL DATA:
CALLED BY: GET-MODULE-COMPLEXITY
           FIND-CONDITIONAL-SITUATIONS
           CHECK-MODULE-START
           ANALYZE-COMPLEXITY

GET-MODULE-COMPLEXITY.
MOVE ZERO TO CONDITIONAL-SITUATION-SM.
PERFORM FIND-CONDITIONAL-SITUATIONS
UNTIL END-OF-FILE
      OR A-CONDITIONAL-SITUATION
      OR AREA-A-OCCUPIED.
IF NOT-END-OF-FILE
  IF AREA-A-OCCUPIED
    PERFORM CHECK-MODULE-START
  ELSE IF A-CONDITIONAL-SITUATION
    PERFORM ANALYZE-COMPLEXITY.
  
```

```
LINE  DEBUG SEQ/NO  A.....10.....
1057* /
1058* *****
1059* * 5.1.1 FIND CONDITIONAL SITUATIONS
1060* *
1061* *
1062* *
1063* * FUNCTION:  SCAN THE COROL SOURCE PROGRAM, LOOKING FOR
1064* * A CONDITIONAL
1065* * SOURCE-WORD
1066* * SOURCE-WORD
1067* * ACCUMULATIVE-COMPLEXITY
1068* * RUNNING-SECTION-TOTAL
1069* * RUNNING-PROGRAM-TOTAL
1070* * TIME-TO-ADD-ONE-TO-COMPLEXITY
1071* * GO-TO-DEPENDING-SW
1072* * GO-TO-COMPLEXITY
1073* * GET-MODULE-COMPLEXITY
1074* * GET-WORD
1075* * SCAN-FOR-GOTO-DEPENDING
1076* *
1077* *****
1078* * FIND-CONDITIONAL-SITUATIONS.
1079* * IF TIME-TO-ADD-ONE-TO-COMPLEXITY
1080* * ADD 1 TO ACCUMULATIVE-COMPLEXITY
1081* * ADD 1 TO RUNNING-SECTION-TOTAL
1082* * ADD 1 TO RUNNING-PROGRAM-TOTAL
1083* * ADD 1 TO RUNNING-SECTION-COUNT
1084* * ADD 1 TO RUNNING-PROGRAM-COUNT
1085* * MOVE ZERO TO ADD-ONE-SW.
1086* * IF SOURCE-WORD = "INVALID"
1087* * MOVE ZERO TO GO-TO-DEPENDING-SW
1088* * PERFORM GET-WORD
1089* * IF NOT-END-OF-FILE
1090* * IF SOURCE-WORD = "KEY"
1091* * ADD 1 TO ACCUMULATIVE-COMPLEXITY
1092* * ADD 1 TO RUNNING-SECTION-TOTAL
1093* * ADD 1 TO RUNNING-PROGRAM-TOTAL.
1094* * IF NOT-END-OF-FILE
1095* * IF SOURCE-WORD = "AT"
1096* * MOVE ZERO TO GO-TO-DEPENDING-SW
1097* * PERFORM GET-WORD
1098* * IF NOT-END-OF-FILE
1099* * IF SOURCE-WORD = "END"
1100* * OR SOURCE-WORD = "END-OF-PAGE"
1101* * ADD 1 TO ACCUMULATIVE-COMPLEXITY
1102* * ADD 1 TO RUNNING-SECTION-TOTAL
1103* * ADD 1 TO RUNNING-PROGRAM-TOTAL.
1104* * IF NOT-END-OF-FILE
1105* * IF SOURCE-WORD = "IF" OR SOURCE-WORD = "UNTIL"
1106* * MOVE ZERO TO GO-TO-DEPENDING-SW
1107* * MOVE 1 TO CONDITIONAL-SITUATION-SW
1108* * ELSE PERFORM SCAN-FOR-GOTO-DEPENDING.
1109* * IF NOT-END-OF-FILE
1110* * PERFORM GET-WORD.
1111* * D DISPLAY ACCUMULATIVE-COMPLEXITY.
1112* * D LINE 15. POSITION 34.
```

LINE DEBUG SEG/NO A.....:.....: 10.....

```

11130 /
11140 *****
11150 * 5.1.1.1 SCAN FOR GOTO DEPENDING *****
11160 *
11170 *
11180 *
11190 *
11200 * FUNCTION: FOR EVERY POSSIBLE GOTO DEPENDING
11210 * SITUATION, ACCUMULATE THE POTENTIAL
11220 * COMPLEXITY. IF THE DEPENDING PHRASE IS
11230 * FOUND, ADD TO COMPLEXITY TOTALS THE
11240 * ACCUMULATED COMPLEXITY FOR THE GOTO
11250 * DEPENDING.
11260 * INPUT: SOURCE-WORD
11270 * ACCUMULATIVE-COMPLEXITY
11280 * RUNNING-SECTION-TOTAL
11290 * RUNNING-PROGRAM-TOTAL
11300 * GOTO-COMPLEXITY
11310 * GOTO-COMPLEXITY
11320 * GOTO-DEPENDING-SW
11330 * ACCUMULATIVE-COMPLEXITY
11340 * LOCAL DATA: NONE
11350 * CALLED BY: FIND-CONDITIONAL-SITUATIONS
11360 * CALLS: NONE
11370 *****
11380 * SCAN-FOR-GOTO-DEPENDING.
11390 * IF SOURCE-WORD = "GOTO"
11400 * MOVE ZERO TO GOTO-COMPLEXITY
11410 * MOVE 1 TO GOTO-DEPENDING-SW
11420 * ELSE IF SOURCE-WORD = "DEPENDING"
11430 * MOVE ZERO TO GOTO-DEPENDING-SW
11440 * ADD GOTO-COMPLEXITY TO ACCUMULATIVE-COMPLEXITY
11450 * ADD GOTO-COMPLEXITY TO RUNNING-SECTION-TOTAL
11460 * ADD GOTO-COMPLEXITY TO RUNNING-PROGRAM-TOTAL
11470 * ELSE IF GOTO-DEPENDING-SITUATION
11480 * AND SOURCE-WORD NOT EQUAL "TO"
11490 * ADD 1 TO GOTO-COMPLEXITY.

```

LINE DEBUG SEQ/NO A...B.....: ID.....

```

1150* /
1151* .....:
1152* .....:
1153* .....:
1154* .....:
1155* .....:
1156* .....:
1157* .....:
1158* .....:
1159* .....:
1160* .....:
1161* .....:
1162* .....:
1163* .....:
1164* .....:
1165* .....:
1166* .....:
1167* .....:
1168* .....:
1169* .....:
1170* .....:
1171* .....:
1172*0403C6 CHECK-MODULE-START.
1173*0403C6 PERFORM SET-PRINT-AVE-N-TOTAL-SW.
1174* .....:
1175* .....:
1176* .....:
1177*0403C8 IF NOT-FIRST-TIME-THRU
1178* .....:
1179* .....:
1180*0403D6 ELSE MOVE 1 TO FIRST-TIME-THRU-SW.
1181*0403DA MOVE SOURCE-WORD TO MODULE-NAME.
1182*0403DC PERFORM GET-WORD.
1183* .....:
1184* .....:
1185* .....:
1186* .....:
1187* .....:
1188* .....:
1189* .....:
1190* .....:
1191* .....:
1192* .....:
1193* .....:
1194* .....:
1195* .....:
1196* .....:
1197*040410 SET-PRINT-AVE-N-TOTAL-SW.
1198*040410 IF ( NOT-FIRST-TIME-THRU AND RUNNING-SECTION-TOTAL > 0)
1199* .....:
1200* .....:
1201* .....:
1202*040428 CHECK-FOR-AVE-N-TOTAL-PRINT.
1203*040428 IF PRINT-AVE-N-TOTAL
1204* .....:

```

```

LINE  DEBUG SEQ/NO  A..... ID.....
12050  /
12060  *
12070  * 5.1.3 ANALYZE COMPLEXITY
12080  *
12090  *
12100  *
12110  * FUNCTION: SCAN A CONDITIONAL, CALCULATING ITS
12120  * COMPLEXITY
12130  * INPUT: NONE
12140  * OUTPUT: NONE
12150  * LOCAL DATA: MORE-CONDITION-SW
12160  * DUNE-SW
12170  * NO-NOT-SW
12180  * RELATIONAL-OPERATOR-SW
12190  * GET-MODULE-COMPLEXITY
12200  * GET-PAST-AND-REMOVE-PARENS
12210  * SCAN-FOR-NOT
12220  * SCAN-QUALIFICATION-N-SUBSCRIPT
12230  * SCAN-FOR-IS-N-NOT
12240  * CHECK-IF-RELATIONAL-OPERATOR
12250  * SCAN-FOR-OR-N-AND
12260  *
12270  *
12280  * ANALYZE-COMPLEXITY.
12290  * MOVE ZERO TO MORE-CONDITION-SW.
12300  * PERFORM SCAN-CONDITIONAL
12310  * UNTIL NO-MORE-CONDITION.
12320  *
12330  * SCAN-CONDITIONAL.
12340  * MOVE ZERO TO DUNE-SW.
12350  * PERFORM GET-PAST-AND-REMOVE-PARENS
12360  * UNTIL DUNE.
12370  * MOVE ZERO TO NO-NOT-SW.
12380  * PERFORM SCAN-FOR-NOT
12390  * UNTIL NO-NOTS.
12400  * PERFORM SCAN-QUALIFICATION-N-SUBSCRIPT.
12410  * PERFORM SCAN-FOR-IS-N-NOT.
12420  * MOVE ZERO TO RELATIONAL-OPERATOR-SW.
12430  * PERFORM CHECK-IF-RELATIONAL-OPERATOR.
12440  * IF NOT-A-RELATIONAL-OPERATOR
12450  * PERFORM CHECK-OUT-CONDITION.
12460  * PERFORM SCAN-FOR-OR-N-AND.

```



```

LINE  DEBUG SEQ/NO A...8.....
1278*  /
1279*  *****
1280*  *
1281*  * 5.1.3.2 SCAN FOR NOT
1282*  *
1283*  *
1284*  *
1285*  * FUNCTION: SCAN FOR NOT, SETTING NO-NOT-SW AS TO
1286*  *          WHETHER ONE IS FOUND OR NOT FOUND
1287*  *          INPUT:  SAVE-WORD
1288*  *          OUTPUT: NO-NOT-SW
1289*  *          LOCAL DATA: DONE-SW
1290*  *          CALLED BY: SCAN-CONDITIONAL
1291*  *          CALLS:  GET-PAST-AND-REMOVE-PARENS
1292*  *          GET-WORD
1293*  *
1294*  *
1295*  * SCAN-FOR-NOT.
1296*  * IF NOT-END-OF-FILE
1297*  *   IF SAVE-WORD = "NOT"
1298*  *     PERFORM GET-WORD
1299*  *     MOVE ZERO TO DONE-SW
1300*  *     PERFORM GET-PAST-AND-REMOVE-PARENS
1301*  *     UNTIL DONE
1302*  *     ELSE MOVE 1 TO NO-NOT-SW
1303*  *     PERFORM GET-WORD
1304*  *     ELSE MOVE 1 TO NO-NOT-SW.

```

LINE DEBUG SEGNO A..... ID.....

```

1304* /
1305* .....
1306* .....
1307* .....
1308* .....
1309* .....
1310* .....
1311* .....
1312* .....
1313* .....
1314* .....
1315* .....
1316* .....
1317* .....
1318* .....
1319* .....
1320* .....
1321* .....
1322* .....
1323* .....
1324* .....
1325*0404E6 .....
1326*0404E6 .....
1327*0404EA .....
1328* .....
1329* .....
1330* .....
1331* .....
1332* .....
1333* .....
1334*04050A .....
1335*04050A .....
1336*04050C .....
1337*04050E .....
1338* .....
1339*040518 .....

*****
5.1.3.3 SCAN QUALIFICATION N SUBSCRIPT
*****
FUNCTION: SCAN ANY QUALIFICATION AND ASSOCIATED
SUBSCRIPTS, SCANNING PAST THE
SUBSCRIPTS, AND PLACING THE
QUALIFICATION ITEMS IN THE TABLE CALLED
QUALIFICATION ENTRIES.
INPUT: SOURCE-WORD
OUTPUT: QUALIFICATION-COUNTER
QUALIFICATION-ENTRIES
SOURCE-WORD
LOCAL DATA: NONE
CALLED BY: SCAN-CONDITIONAL
CHECK-IF-RELATIONAL-OPERATOR
CALLS: GET-WORD
*****
SCAN-QUALIFICATION-N-SUBSCRIPT.
MOVE ZERO TO QUALIFICATION-COUNTER.
IF NOT-END-OF-FILE
PERFORM SCAN-QUALIFICATION
UNTIL SOURCE-WORD NOT = "IN"
AND SOURCE-WORD NOT = "OF"
IF SOURCE-WORD-ITEM (1) = "("
PERFORM GET-WORD.
*****
SCAN-QUALIFICATION.
PERFORM GET-WORD.
ADD 1 TO QUALIFICATION-COUNTER.
MOVE SOURCE-WORD TO
QUALIFICATION-ITEM (QUALIFICATION-COUNTER).
PERFORM GET-WORD.
*****

```


LINE DEMO SEQ/NO A-B-C-D-E-F-G-H-I-J-K-L-M-N-O-P-Q-R-S-T-U-V-W-X-Y-Z

```

1340* /
1341* .....
1342* .....
1343* .....
1344* .....
1345* .....
1346* .....
1347* .....
1348* .....
1349* .....
1350* .....
1351* .....
1352* .....
1353* .....
1354*04051C .....
1355*04051C .....
1356* .....
1357* .....
1358*04052A .....
1359* .....
1360* .....

S-I-3-4 SCAN FOR IS N NOT

FUNCTION: SCAN PAST IS AND NOT IF THEY ARE FOUND.
INPUT: SOURCE-WORD
OUTPUT: SOURCE-WORD
LOCAL DATA: NONE
CALLED BY: SCAN-CONDITIONAL
CALLS: GET-WORD

SCAN-FOR-IS-N-NOT.
IF NOT-END-OF-FILE
  IF SOURCE-WORD = "IS"
    PERFORM GET-WORD.
  IF NOT-END-OF-FILE
    IF SOURCE-WORD = "NOT"
      PERFORM GET-WORD.
  
```

MCB-IMDS C080L COMPILATION VERSION 05.12.00 01/03/14 10:10:00 SI-MCCABES DO-DMCCABES

```

LINE  DEBUG SEQ/NO  A.....B..... ID.....
13610  /
13620  .....
13630  .....
13640  .....
13650  .....
13660  .....
13670  .....
13680  .....
13690  .....
13700  .....
13710  .....
13720  .....
13730  .....
13740  .....
13750  .....
13760  .....
13770  .....
13780  .....
13790  .....
13800  .....

          5.1.3.5  SCAN FOR OR N AND

          FUNCTION:  SCAN FOR THE CONNECTORS 'OR' AND 'AND',
                     WHICH INDICATE COMPOUND CONDITIONALS
          INPUT:      SOURCE-WORD
          OUTPUT:     SOURCE-WORD
                     MORE-CONDITION-SW
          LOCAL DATA:  NONE
          CALLED BY:   SCAN-CONDITIONAL
          CALLS:       GET-WORD

          SCAN-FOR-OR-N-AND.
          IF SOURCE-WORD = "OR" OR SOURCE-WORD = "AND"
             PERFORM GET-WORD
          ELSE MOVE 1 TO MORE-CONDITION-SW.

```


NCR-TRUS COBOL COMPILATION VERSION 05.12.03 81/03/14 18:18:00 SI-MCCABES BU-ORCCABES

```

LINE  DEBUC SEQ/NO A...B..... ID.....
1-220 /
1-230
1-240
1-250
1-260
1-270
1-280
1-290
1-300
1-310
1-320
1-330
1-340
1-350
1-360
1-370
1-380
1-390
1-400
1-410
1-420
1-430
1-440
1-450
1-460
1-470
1-480
1-490
1-500
1-510
1-520
1-530
1-540
1-550
1-560
1-570
1-580
1-590
1-600
1-610
1-620
1-630
1-640
1-650
1-660
1-670
1-680
1-690
1-700
1-710
1-720

```

5.1.3.7 CHECK OUT CONDITION

FUNCTION: DETERMINE THE COMPLEXITY OF THE
CONDITIONAL BEING SCANNED.

INPUT: ACCUMULATIVE-COMPLEXITY
RUNNING-SECTION-TOTAL
RUNNING-PROGRAM-TOTAL

OUTPUT: ACCUMULATIVE-COMPLEXITY
RUNNING-SECTION-TOTAL
RUNNING-PROGRAM-TOTAL

LOCAL DATA: NONE

CALLED BY: SCAN-CONDITIONAL

CALLS: CO-THRU-CONDITION-NAMES

CHECK-OUT-CONDITION.

IF CLASS-CONDITION

ADD 1 TO ACCUMULATIVE-COMPLEXITY

ADD 1 TO RUNNING-SECTION-TOTAL

ADD 1 TO RUNNING-PROGRAM-TOTAL

DISPLAY ACCUMULATIVE-COMPLEXITY,

LINE 15, POSITION 34

ELSE PERFORM LOOK-FOR-CONDITION-NAME.

LOOK-FOR-CONDITION-NAME.

MOVE ZERO TO CONDITION-FIND-FLAG.

MOVE ZERO TO SINE-KEY.

PERFORM GO-THRU-CONDITION-NAMES

UNTIL SINE-KEY EQUAL NUMBER-OF-CONDITION-NAMES

OR CONDITION-NAME-FOUND.

IF CONDITION-NAME-FOUND

ADD COMPLEXITY-VALUE TO ACCUMULATIVE-COMPLEXITY

ADD COMPLEXITY-VALUE TO RUNNING-SECTION-TOTAL

ADD COMPLEXITY-VALUE TO RUNNING-PROGRAM-TOTAL

ELSE DISPLAY "TRIED TO SEARCH FOR",

LINE 21, POSITION 1,

"A NON-EXISTENT CONDITION-NAME --> ",

LINE 21, POSITION 21,

SAVE-WORD.

LINE 21, POSITION 56,

"IN LINE NUMBER",

LINE 22, POSITION 5,

LINE-NUMBER-DISPLAY,

LINE 22, POSITION 20,

IF CONDITION-NAME-FOUND

DISPLAY ACCUMULATIVE-COMPLEXITY,

LINE 15, POSITION 34.

NON-ANSI

NON-ANSI

NON-ANSI

NON-ANSI

NON-ANSI

NCR-IMDS COBOL COMPILATION VERSION 05.12.00 01/03/14 10:10:00 SI-MCCABES DU-ORCADES

LINE DEBUC SEQ/NO A...B..... ID.....

```

14730 /
14740 *
14750 *
14760 *
14770 *
14780 *
14790 *
14800 *
14810 *
14820 *
14830 *
14840 *
14850 *
14860 *
14870 *
14880 *
14890 *
14900 *
14910 *
14920 *
14930 *
14940 *
14950 *
14960 *
14970 040656
14980 040656
14990 040658
15000 *
15010 *
15020 040660
15030 *
15040 *
15050 *
15060 *
15070 *
15080 *
15090 040659
15100 040659
15110 040659
15120 040662
15130 *
15140 *
15150 040685
15160 *
15170 *
15180 0406C2
15190 0406C2
15200 0406D0
15210 *
15220 *
15230 0406E3
15240 *

```

5.1.3.7.1 GO THRU CONDITION NAMES
 FUNCTION: LOOK THROUGH THE SIRE FILE FOR AN ENTRY
 THAT MATCHES THE CONDITION-NAME
 CURRENTLY POINTED TO IN THE SOURCE
 PROGRAM.
 INPUTS:
 SIRE-KEY
 SIRE-FILE
 SAVE-WORD
 OUTPUTS:
 QUALIFICATION-COUNTER
 SIRE-KEY
 CONDITION-FIND-FLAG
 LOCAL DATA:
 CONDITION-NAME-ENTRY
 SUBSCRIPT-TEMP
 SIRE-TABLE-SUBSCRIPT
 NAME-KEY
 LOOK-FOR-CONDITION-NAME
 CALLED BY:
 CALLS:
 NONE

GO-THRU-CONDITION-NAMES.
 ADD 1 TO SIRE-KEY.
 READ SIRE-FILE RECORD INTO CONDITION-NAME-ENTRY.
 INVALID KEY DISPLAY "SIRE-FILE SEARCH ERROR"
 STOP RUN.
 IF SAVE-WORD = CONDITION-NAME
 IF A-SWITCH
 MOVE 1 TO CONDITION-FIND-FLAG
 ELSE IF QUALIFICATION-COUNTER > ZERO
 PERFORM COMPARE-QUALIFICATION-TO-SIRES
 ELSE MOVE 1 TO CONDITION-FIND-FLAG.
 COMPARE-QUALIFICATION-TO-SIRES.
 MOVE 1 TO SUBSCRIPT-TEMP.
 MOVE 1 TO SIRE-TABLE-SUBSCRIPT.
 PERFORM COMPARE-SIRES
 UNTIL SIRE-TABLE-SUBSCRIPT > TABLE-SIZE
 OR SUBSCRIPT-TEMP > QUALIFICATION-COUNTER.
 IF SUBSCRIPT-TEMP IS GREATER THAN QUALIFICATION-COUNTER
 MOVE 1 TO CONDITION-FIND-FLAG.
 COMPARE-SIRES.
 MOVE SIRE-NAME-POINTER (SIRE-TABLE-SUBSCRIPT) TO NAME-KEY.
 READ NAME-FILE RECORD INTO SIRE-NAME
 INVALID KEY DISPLAY "NAME-FILE READ ERROR"
 STOP RUN.
 IF QUALIFICATION-ITEM (SUBSCRIPT-TEMP) = SIRE-NAME
 ADD 1 TO SIRE-TABLE-SUBSCRIPT.

```

LINE  DEBUG SEQ/NO  A...8..... ID.....
15259  /.....
15260  .....
15270  .....
15280  .....
15290  .....
15300  .....
15310  .....
15320  .....
15330  .....
15340  .....
15350  .....
15360  .....
15370  .....
15380  .....
15390  .....
15400  .....
15410  .....
15420  .....
15430  .....
15440  .....
15450  .....
15460  .....
15470  .....
15480  .....
15490  .....
15500  .....
15510  .....
15520  .....
15530  .....
15540  .....
15550  .....
15560  .....
15570  .....
15580  .....
15590  .....
15600  .....
15610  .....
15620  .....
15630  .....
15640  .....
15650  .....
15660  .....
15670  .....
15680  .....
15690  .....
15700  .....
15710  .....

/.....
* 6 PRINT ROUTINES
.....
PRINT-SECTION-NAME-NUMBER.
PERFORM GET-TO-TOP-OF-PAGE.
MOVE MODULE-NAME TO SECTION-NAME-1.
WRITE PRINT-RECORD FROM PRINT-LINE-1 AFTER ADVANCING 3 LINES
AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.
DISPLAY SPACES, LINE 11, SIZE 80.
DISPLAY "SECTION NAME - ",
LINE 11, POSITION 15,
MODULE-NAME.
LINE 11, POSITION 30,
SECTION-NUMBER.
LINE 11, POSITION 61.
*
PRINT-SECTION-NAME.
PERFORM GET-TO-TOP-OF-PAGE.
MOVE MODULE-NAME TO SECTION-NAME-2.
WRITE PRINT-RECORD FROM PRINT-LINE-2 AFTER ADVANCING 3 LINES
AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.
DISPLAY SPACES, LINE 11, SIZE 80.
DISPLAY "SECTION NAME - ",
LINE 11, POSITION 15,
MODULE-NAME.
LINE 11, POSITION 30.
*
PRINT-MODULE-NAME.
WRITE PRINT-RECORD FROM PRINT-LINE-3 AFTER ADVANCING 2 LINES
AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.
DISPLAY SPACES, LINE 13, SIZE 80.
DISPLAY "PARAGRAPH NAME - ",
LINE 13, POSITION 15,
MODULE-NAME.
LINE 13, POSITION 32.
*
PRINT-COMPLEXITY.
MOVE ACCUMULATIVE-COMPLEXITY TO COMPLEXITY.
WRITE PRINT-RECORD FROM PRINT-LINE-4 AFTER ADVANCING 1 LINES
AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.
MOVE ZERO TO ACCUMULATIVE-COMPLEXITY.
MOVE 1 TO ADD-ONE-SM.
DISPLAY ACCUMULATIVE-COMPLEXITY.
LINE 15, POSITION 54.

```

MCR-1405 COBOL COMPILATION VERSION 05.12.00 81/03/14 16:18:00 S1-MCCABES BU-MCCABES

```

LINE DL=JG SEQ#40 A...B..... ID.....
1572* /
1573*040750 PRINT-SECTION-TOTAL-N-AVERAGE. 1
1574*040750 MOVE RUNNING-SECTION-TOTAL TO RUNNING-SECTION-TOTAL-OUT. 1
1575*040755 WRITE PRINT-RECORD FROM PRINT-LINE-5 AFTER ADVANCING 4 LINES 1
1576* AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE. 1
1577*040753 DIVIDE RUNNING-SECTION-TOTAL 1
1578* BY RUNNING-SECTION-COUNT 1
1579* GIVING SECTION-AVERAGE ROUNDED. 1
1580*040760 MOVE SECTION-AVERAGE TO SECTION-AVERAGE-OUT. 1
1581*040773 WRITE PRINT-RECORD FROM PRINT-LINE-6 AFTER ADVANCING 2 LINES 1
1582* AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE. 1
1583*040781 MOVE ZEROS TO RUNNING-SECTION-TOTAL. 1
1584* RUNNING-SECTION-COUNT. 1
1585* 1
1586*040789 PRINT-PROGRAM-TOTAL-N-AVERAGE. 1
1587*040789 PERFORM GET-TO-TOP-OF-PAGE. 1
1588*04078C MOVE RUNNING-PROGRAM-TOTAL TO RUNNING-PROGRAM-TOTAL-OUT. 1
1589*040791 WRITE PRINT-RECORD FROM PRINT-LINE-7 AFTER ADVANCING 4 LINES 1
1590* AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE. 1
1591*04079F DIVIDE RUNNING-PROGRAM-TOTAL 1
1592* BY RUNNING-PROGRAM-COUNT 1
1593* GIVING PROGRAM-AVERAGE ROUNDED. 1
1594*0407A9 MOVE PROGRAM-AVERAGE TO PROGRAM-AVERAGE-OUT. 1
1595*0407AF WRITE PRINT-RECORD FROM PRINT-LINE-8 AFTER ADVANCING 2 LINES 1
1596* AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE. 1
1597* ***** END OF FILE***** 1
2ZZZZZ END PROGRAM.

```

ADDRESS SIZE DEBUG ORDER TYPE NAME

*** LOCAL ***

0030	480	0	ANS	0	FILE	FIXED-INPUT
0148	80	0	ANS	0	ALPHANUMERIC	INPUT-LINE
0048	512	0	ANS	0	FILE	VARIABLE-INPUT
0168	80	0	ANS	0	ALPHANUMERIC	INPUT-LINE
0070	132	0	ANS	0	FILE	PRINT-FILE
0188	132	0	ANS	0	ALPHANUMERIC	PRINT-RECORD
0008	351	0	GRP	0	FILE	SIRE-FILE
023C	30	0	ANS	0	GROUP	SIRE-RECORD
023C	30	0	ANS	0	ALPHANUMERIC	S-COND-NAME
025A	6	1	GRP	1	GROUP	S-SIRE-TABLE
025A	4	1	NSU	1	NUMERIC	S-SIRE-NAME-POINTER
025E	2	1	ANS	1	ALPHANUMERIC	S-SIRE-LEVEL
0386	18	0	NSU	0	NUMERIC	S-COMPLEXITY-VALUE
0378	1	0	NSU	0	NUMERIC	S-SWITCH-FLAG
0379	2	0	NSU	0	NUMERIC	S-TABLE-SIZE
0120	510	0	ANS	0	FILE	NAME-FILE
039C	30	0	ANS	0	ALPHANUMERIC	NAME-RECORD
03C8	80	0	GRP	0	GROUP	SOURCE-LINE
03C8	6	0	ANS	0	ALPHANUMERIC	SEQUENCE-NUMBER
03CE	1	0	ANS	0	ALPHANUMERIC	COMMENT-FIELD
03CF	65	0	ANS	0	CONDICTION-NAME	COMMENT-OR-DEBUG
0410	8	0	ANS	0	CONDICTION-NAME	LINE-CONTINUATION
03C8	80	0	GRP	0	ALPHANUMERIC	TEXT-LINE
03C8	1	0	GRP	0	ALPHANUMERIC	ID-FIELD
03C8	1	0	GRP	0	ALPHANUMERIC	SOURCE-LINE-REDEFINED
03C8	1	0	GRP	0	ALPHANUMERIC	SOURCE-ITEMS
03C8	1	0	GRP	0	ALPHANUMERIC	SOURCE-LINE-ITEM
0418	351	0	GRP	0	GROUP	CONDICTION-NAME-ENTRY
0418	30	0	ANS	0	ALPHANUMERIC	CONDICTION-NAME
0436	6	1	GRP	1	GROUP	SIRE-TABLE
0436	4	1	NSU	1	NUMERIC	SIRE-NAME-POINTER
043A	2	1	ANS	1	ALPHANUMERIC	SIRE-LEVEL
0562	18	0	NSU	0	NUMERIC	COMPLEXITY-VALUE
0574	1	0	NSU	0	NUMERIC	SWITCH-FLAG
0575	2	0	NSU	0	CONDICTION-NAME	IS-A-SWITCH
0575	2	0	NSU	0	CONDICTION-NAME	NOT-A-SWITCH
0575	2	0	NSU	0	NUMERIC	TABLE-SIZE
0578	120	0	ANS	0	ALPHANUMERIC	SOURCE-WORD
0578	120	0	ANS	0	ALPHANUMERIC	A-LEVEL-NUMBER
0578	120	0	ANS	0	ALPHANUMERIC	RELATIONAL-OPERATOR
0578	120	0	ANS	0	ALPHANUMERIC	A-SWITCH
0578	120	0	ANS	0	ALPHANUMERIC	CLASS-CONDITION
0578	120	0	GRP	0	GROUP	SOURCE-WORD-REDEFINED
0578	120	0	GRP	0	GROUP	SOURCE-WORDS

ADDRESS	SIZE	DEBUG	UNDER	TYPE	NAME
0578	1	ANS	1	ALPHANUMERIC	SOURCE-WORD-ITEM
05F0	30	ANS	0	ALPHANUMERIC	SAVE-WORD
060E	1500	GRP	0	GROUP	QUALIFICATION-ENTRIES
060E	30	GRP	1	GROUP	QUALIFICATION-TABLE
060E	30	ANS	1	ALPHANUMERIC	QUALIFICATION-ITEM
08E4	6	GRP	0	GROUP	THE-TIME
08E4	2	NSU	0	NUMERIC	HOURS
08E4	2	NSU	0	NUMERIC	MINUTES
08E4	2	NSU	0	NUMERIC	SECONDS
08F0	121	GRP	0	GROUP	HEADER
0C24	30	ANS	0	ALPHANUMERIC	PROGRAM-NAME
0C47	8	NSE	0	NUMERIC EDITED	DATE-OUT
0C54	2	NSU	0	NUMERIC	HOURS-OUT
0C57	2	NSU	0	NUMERIC	MINUTES-OUT
0C5A	2	NSU	0	NUMERIC	SECONDS-OUT
0C66	3	ANS	0	ALPHANUMERIC	PAGE-NUMBER
0C6A	73	GRP	0	GROUP	PRINT-LINE-1
0C80	30	ANS	0	ALPHANUMERIC	SECTION-NAME-1
0C81	2	ANS	0	ALPHANUMERIC	SECTION-NUMBER
0C84	54	GRP	0	GROUP	PRINT-LINE-2
0CCC	30	ANS	0	ALPHANUMERIC	SECTION-NAME-2
0CEA	61	GRP	0	GROUP	PRINT-LINE-3
0D09	30	ANS	0	ALPHANUMERIC	MODULE-NAME
0D28	46	GRP	0	GROUP	PRINT-LINE-4
0D44	18	NSE	0	NUMERIC EDITED	COMPLEXITY
0D56	64	GRP	0	GROUP	PRINT-LINE-5
0D84	18	NSE	0	NUMERIC EDITED	RUNNING-SECTION-TOTAL-OUT
0D96	67	GRP	0	GROUP	PRINT-LINE-6
0DC6	19	NSE	0	NUMERIC EDITED	SECTION-AVERAGE-OUT
0DDA	64	GRP	0	GROUP	PRINT-LINE-7
0E03	18	NSE	0	NUMERIC EDITED	RUNNING-PROGRAM-TOTAL-OUT
0E1A	67	GRP	0	GROUP	PRINT-LINE-8
0E4A	19	NSE	0	NUMERIC EDITED	PROGRAM-AVERAGE-OUT
0E50	6	NSU	0	NUMERIC	THE-DATE
0E63	18	NSU	0	NUMERIC	ACCUMULATIVE-COMPLEXITY
0E75	18	NSU	0	NUMERIC	GO-TO-COMPLEXITY
0E87	18	NSU	0	NUMERIC	QUALIFICATION-COUNTER
0E99	3	NSU	0	NUMERIC	PAGE-COUNT

ADDRESS	SIZE	DEBUG	UNDEF	TYPE	NAME
0E9C	18	NSU	0	NUMERIC	NUMBER-OF-CONDITION-NAMES
0EAE	18	NSU	0	NUMERIC	SIRE-KEY
0ECO	18	NSU	0	NUMERIC	NAME-KEY
0ED2	18	NSU	0	NUMERIC	PTR
0EE4	18	NSU	0	NUMERIC	START-OF-WORD
0EF6	18	NSU	0	NUMERIC	END-OF-WORD
0F08	18	NSU	0	NUMERIC	WORD-SIZE
0F1A	18	NSU	0	NUMERIC	SOURCE-PTR
0F2C	18	NSU	0	NUMERIC	PTR-TEMP
0F3E	2	ANS	0	ALPHANUMERIC	DELIMITER-TEMP
0F40	120	ANS	0	ALPHANUMERIC	SOURCE-WORD-TEMP
0F80	2	ANS	0	ALPHANUMERIC	SOURCE-STOP
0F8A	2	NSU	0	NUMERIC	ID-1
0F8C	2	NSU	0	NUMERIC	SIRE-TABLE-SUBSCRIPT
0F8E	18	NSU	0	NUMERIC	SUBSCRIPT-TEMP
0FD0	2	ANS	0	ALPHANUMERIC	CURRENT-LEVEL
0FD2	2	ANS	0	ALPHANUMERIC	LAST-LEVEL
0FD4	18	NSU	0	NUMERIC	RUNNING-SECTION-COUNT
0FE6	18	NSU	0	NUMERIC	RUNNING-PROGRAM-COUNT
0FF8	18	NSU	0	NUMERIC	RUNNING-SECTION-TOTAL
100A	18	NSU	0	NUMERIC	RUNNING-PROGRAM-TOTAL
101C	18	NSU	0	NUMERIC	SECTION-AVERAGE
102E	18	NSU	0	NUMERIC	PROGRAM-AVERAGE
1040	30	ANS	0	ALPHANUMERIC	SIRE-NAME FILLER-SIRE
105E	1	NSU	0	NUMERIC	FILE-FLAG FIXED-LENGTH-FILE VARIABLE-LENGTH-FILE
105F	1	NSU	0	NUMERIC	EUF-FLAG NOT-END-OF-FILE END-OF-FILE

ADDRESS	SIZE	DEBUG	ORDER	TYPE	NAME
1060	1	NSU	0	NUMERIC CONDITION-NAME	AREA-A-FLAG AREA-A-OCCUPIED
1061	1	NSU	0	NUMERIC CONDITION-NAME	CONDITION-FIND-FLAG CONDITION-NAME-FOUND
1062	1	NSU	0	NUMERIC CONDITION-NAME	DONL-SW DONE
1063	1	NSU	0	NUMERIC CONDITION-NAME	CONSUME-SW CONSUME-DONE
1064	1	NSU	0	NUMERIC CONDITION-NAME	QUOTE-SW BALANCED-QUOTE
1065	1	NSU	0	NUMERIC CONDITION-NAME	CONDITIONAL-SITUATION-SW A-CONDITIONAL-SITUATION
1066	1	NSU	0	NUMERIC CONDITION-NAME	NO-NOT-SW NO-NOTS
1067	1	NSU	0	NUMERIC CONDITION-NAME	MORE-CONDITION-SW NO-MORE-CONDITION
1068	1	NSU	0	NUMERIC CONDITION-NAME	RELATIONAL-OPERATOR-SW NOT-A-RELATIONAL-OPERATOR
1069	1	NSU	0	NUMERIC CONDITION-NAME	GO-TO-DEPENDING-SW GO-TO-DEPENDING-SITUATION
106A	1	NSU	0	NUMERIC CONDITION-NAME	FIRST-TIME-THRU-SW NOT-FIRST-TIME-THRU
106B	4	NSU	0	NUMERIC	LINE-NUMBER
106F	4	NSF	0	NUMERIC EDITED	LINE-NUMBER-DISPLAY
1073	1	NSU	0	NUMERIC CONDITION-NAME	ADD-ONE-SW TIME-TO-ADD-ONE-TO-COMPLEXITY
1074	1	NSU	0	NUMERIC CONDITION-NAME	PRINT-AVE-N-TOTAL-SW PRINT-AVE-N-TOTAL

REQUIRED RUNTIME ELEMENTS:

NUCLEUS
ACCEPT/DISPLAY
LINAGE
MULTIPLY/DIVIDE
STRING

LOCAL BYTE SIZE	6702
GLOBAL BYTE SIZE	1086
TOTAL BYTE SIZE	7788

WCK-1M05 C300L COMPILATION VERSION 05.12.00 01/03/14 10:18:00 ST-MCCABES 80-QMCCABES PAGE 050

ADDRESS SIZE DEBUG ORDER TYPE NAME

NO ERRORS

NO WARNINGS

APPENDIX F

Halstead's Tool Source Code

The following listing is the COBOL source code for the tool to evaluate Halstead's Software Science Metrics.

```

LINE  DEBU  SCD/ND  A.....: ID.....
1*  IDENTIFICATION DIVISION.
2*  PROGRAM-ID.  HALSTEAD5-MEASURE.
3*  AUTHOR.  RANDALL R. HEALS.
4*  DATE-WRITTEN.  00/07/14.
5*  DATE-COMPILED.  01/03/14.
6*  ENVIRONMENT DIVISION.
7*  CONFIGURATION SECTION.
8*  SOURCE-COMPUTER.  NCR-1405.
9*  OBJECT-COMPUTER.  NCR-1405.
10* .....:
11* .....:
12* .....:
13* .....:
14* .....:
15* .....:
16* .....:
17* .....:
18* .....:

```

NCR-1405 COBOL COMPILATION VERSION 09.12.00 81/03/14 10:10:35 SI-HALSTEAD\$ 80-DHALSTEAD\$

```

LINE  DEBUG SEQ/NO A...B..... ID.....
1-0  / INPUT-OUTPUT SECTION.
20-  *
21-  *
22-  * FILE-CONTROL.
23-  *
24-  * SELECT FIXED-INPUT ASSIGN TO DISC.
25-  * SELECT VARIABLE-INPUT ASSIGN TO DISC.
26-  * SELECT PRINT-FILE ASSIGN TO PRINTER.
27-  * SELECT OPERAND-FILE ASSIGN TO DISC
28-  * ORGANIZATION IS RELATIVE;
29-  * ACCESS MODE IS RANDOM;
30-  * RELATIVE KEY IS OPERAND-KEY.
31-  * SELECT LINK-FILE ASSIGN TO DISC
32-  * ORGANIZATION IS RELATIVE;
33-  * ACCESS MODE IS RANDOM;
34-  * RELATIVE KEY IS LINK-KEY.
35-  *
36-  * I-O-CONTROL:
      SAME RECORD AREA FOR FIXED-INPUT, VARIABLE-INPUT.

```

```

Line  1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612 2613 2614 2615 2616 2617 2618 2619 2620 2621 2622 2623 2624 2625 2626 2627 2628 2629 2630 2631 2632 2633 2634 2635 2636 2637 2638 2639 2640 2641 2642 2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664 2665 2666 2667 2668 2669 2670 2671 2672 2673 2674 2675 2676 2677 2678 2679 2680 2681 2682 2683 2684 2685 2686 2687 2688 2689 2690 2691 2692 2693 2694 2695 2696 2697 2698 2699 2700 2701 2702 2703 2704 2705 2706 2707 2708 2709 2710 2711 2712 2713 2714 2715 2716 2717 2718 2719 2720 2721 2722 2723 2724 2725 2726 2727 2728 2729 2730 2731 2732 2733 2734 2735 2736 2737 2738 2739 2740 2741 2742 2743 2744 2745 2746 2747 2748 2749 2750 2751 2752 2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2768 2769 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2783 2784 2785 2786 2787 2788 2789 2790 2791 2792 2793 2794 2795 2796 2797 2798 2799 2800 2801 2802 2803 2804 2805 2806 2807 2808 2809 2810 2811 2812 2813 2814 2815 2816 2817 2818 2819 2820 2821 2822 2823 2824 2825 2826 2827 2828 2829 2830 2831 2832 2833 2834 2835 2836 2837 2838 2839 2840 2841 2842 2843 2844 2845 2846 2847 2848 2849 2850 2851 2852 2853 2854 2855 2856 2857 2858 2859 2860 2861 2862 2863 2864 2865 2866 2867 2868 2869 2870 2871 2872 2873 2874 2875 2876 2877 2878 2879 2880 2881 2882 2883 2884 2885 2886 2887 2888 2889 2890 2891 2892 2893 2894 2895 2896 2897 2898 2899 2900 2901 2902 2903 2904 2905 2906 2907 2908 2909 2910 2911 2912 2913 2914 2915 2916 2917 2918 2919 2920 2921 2922 2923 2924 2925 2926 2927 2928 2929 2930 2931 2932 2933 2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944 2945 2946 2947 2948 2949 2950 2951 2952 2953 2954 2955 2956 2957 2958 2959 2960 2961 2962 2963 2964 2965 2966 2967 2968 2969 2970 2971 2972 2973 2974 2975 2976 2977 2978 2979 2980 2981 2982 2983 2984 2985 2986 2987 2988 2989 2990 2991 2992 2993 2994 2995 2996 2997 2998 2999 3000 3001 3002 3003 3004 3005 3006 3007 3008 3009 3010 3011 3012 3013 3014 3015 3016 3017 3018 3019 3020 3021 3022 3023 3024 3025 3026 3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3038 3039 3040 3041 3042 3043 3044 3045 3046 3047 3048 3049 3050 3051 3052 3053 3054 3055 3056 3057 3058 3059 3060 3061 3062 3063 3064 3065 3066 3067 3068 3069 3070 3071 3072 3073 3074 3075 3076 3077 3078 3079 3080 3081 3082 3083 3084 3085 3086 3087 3088 3089 3090 3091 3092 3093 3094 3095 3096 3097 3098 3099 3100 3101 3102 3103 3104 3105 3106 3107 3108 3109 3110 3111 3112 3113 3114 3115 3116 3117 3118 3119 3120 3121 3122 3123 3124 3125 3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138 3139 3140 3141 3142 3143 3144 3145 3146 3147 3148 3149 3150 3151 3152 3153 3154 3155 3156 3157 3158 3159 3160 3161 3162 3163 3164 3165 3166 3167 3168 3169 3170 3171 3172 3173 3174 3175 3176 3177 3178 3179 3180 3181 3182 3183 3184 3185 3186 3187 3188 3189 3190 3191 3192 3193 3194 3195 3196 3197 3198 3199 3200 3201 3202 3203 3204 3205 3206 3207 3208 3209 3210 3211 3212 3213 3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3245 3246 3247 3248 3249 3250 3251 3252 3253 3254 3255 3256 3257 3258 3259 3260 3261 3262 3263 3264 3265 3266 3267 3268 3269 3270 3271 3272 3273 3274 3275 3276 3277 3278 3279 3280 3281 3282 3283 3284 3285 3286 3287 3288 3289 3290 3291 3292 3293 3294 3295 3296 3297 3298 3299 3300 3301 3302 3303 3304 3305 3306 3307 3308 3309 3310 3311 3312 3313 3314 3315 3316 3317 3318 3319 3320 3321 3322 3323 3324 3325 3326 3327 3328 3329 3330 3331 3332 3333 3334 3335 3336 3337 3338 3339 3340 3341 3342 3343 3344 3345 3346 3347 3348 3349 3350 3351 3352 3353 3354 3355 3356 3357 3358 3359 3360 3361 3362 3363 3364 3365 3366 3367 3368 3369 3370 3371 3372 3373 3374 3375 3376 3377 3378 3379 3380 3381 3382 3383 3384 3385 3386 3387 3388 3389 3390 3391 3392 3393 3394 3395 3396 3397 3398 3399 3400 3401 3402 3403 3404 3405 3406 3407 3408 3409 3410 3411 3412 3413 3414 3415 3416 3417 3418 3419 3420 3421 3422 3423 3424 3425 3426 3427 3428 3429 3430 3431 3432 3433 3434 3435 3436 3437 3438 3439 3440 3441 3442 3443 3444 3445 3446 3447 3448 3449 3450
```



```

LINE  DEBUG REQ/MU 4..... ID.....
370  /
380  DATA DIVISION.
390  FILE SECTION.
400  FD  FIXLU-INPUT
410  BLOCK CONTAINS 6 RECORDS
420  RECORD CONTAINS 80 CHARACTERS
430  LABEL RECORDS ARE STANDARD
440  VALUE OF FILE-ID IS "1".
450  01  INPUT-LINE PIC X(80).
460  *
470  FD  VARIABLE-INPUT
480  BLOCK CONTAINS 512 CHARACTERS
490  RECORD VARYING 6 TO 80 CHARACTERS DEPENDING ON ID-1
500  LABEL RECORDS ARE STANDARD
510  VALUE OF FILE-ID IS "1".
520  01  INPUT-LINE PIC X(80).
530  *
540  FD  PRINT-FILE
550  BLOCK CONTAINS 1 RECORDS
560  RECORD CONTAINS 132 CHARACTERS
570  LABEL RECORDS ARE OMITTED
580  LINAGE IS 50 LINES
590  LINES AT TOP 3,
600  LINES AT BOTTOM 5.
610  01  PRINT-RECORD PIC X(132).
620  *
630  FD  OPERAND-FILE
640  BLOCK CONTAINS 3 RECORDS
650  LABEL RECORDS ARE STANDARD
660  VALUE OF FILE-ID IS "OPERAND".
670  01  OPERAND-RECORD.
680  05  A-OPERAND-NAME PIC X(122).
690  05  OPERAND-REFERENCE-COUNTER PIC 9(18).
700  *
710  FD  LINK-FILE
720  BLOCK CONTAINS 14 RECORDS
730  LABEL RECORDS ARE STANDARD
740  VALUE OF FILE-ID IS "LINK".
750  01  OPERAND-LINK-RECORD.
760  05  OPERAND-POINTER PIC 9(18).

```


LINE DEBUC SEQ/NO A...B..... ID.....

```

180* / 01 PRINT-LINE-4.
181* 05 FILLER PIC X(28) VALUE IS SPACES.
182* 05 FILLER PIC X(30)
183* 05 FILLER VALUE IS "TOTAL FREQUENCY OF OPERATORS".
184* 05 FILLER PIC X(15) VALUE IS SPACES.
185* 05 FILLER PIC X(12) VALUE IS "F1".
186* 05 FILLER PIC X(33) VALUE IS SPACES.
187* 05 OUTPUT-VALUE-4 PIC Z(17)9.
188*
189*
190* 01 PRINT-LINE-5.
191* 05 FILLER PIC X(26) VALUE IS SPACES.
192* 05 FILLER PIC X(27)
193* 05 FILLER VALUE IS "TOTAL FREQUENCY OF OPERANDS".
194* 05 FILLER PIC X(18) VALUE IS SPACES.
195* 05 FILLER PIC X(12) VALUE IS "F2".
196* 05 FILLER PIC X(33) VALUE IS SPACES.
197* 05 OUTPUT-VALUE-5 PIC Z(17)9.
198*
199*
200* 01 PRINT-LINE-6.
201* 05 FILLER PIC X(15) VALUE IS SPACES.
202* 05 FILLER PIC X(18)
203* 05 FILLER VALUE IS "COMPUTED MEASURES".
204* 05 FILLER PIC X(15) VALUE IS SPACES.
205* 05 FILLER PIC X(18)
206* 05 FILLER VALUE IS "PROGRAM VOCABULARY".
207* 05 FILLER PIC X(17) VALUE IS SPACES.
208* 05 FILLER PIC X(5) VALUE IS "N1+N2".
209*
210* 05 FILLER PIC X(30) VALUE IS SPACES.
211* 05 OUTPUT-VALUE-6 PIC Z(17)9.
212*
213* 01 PRINT-LINE-7.
214* 05 FILLER PIC X(28) VALUE IS SPACES.
215* 05 FILLER PIC X(14)
216* 05 FILLER VALUE IS "PROGRAM LENGTH".
217* 05 FILLER PIC X(21) VALUE IS SPACES.
218* 05 FILLER PIC X(15) VALUE IS "F1+F2".
219* 05 FILLER PIC X(30) VALUE IS SPACES.
220* 05 OUTPUT-VALUE-7 PIC Z(17)9.
221*
222* 01 PRINT-LINE-8.
223* 05 FILLER PIC X(28) VALUE IS SPACES.
224* 05 FILLER PIC X(24)
225* 05 FILLER VALUE IS "ESTIMATED PROGRAM LENGTH".
226* 05 FILLER PIC X(11) VALUE IS SPACES.
227* 05 FILLER PIC X(30)
228* 05 FILLER VALUE IS "N0 = (N1LOG2(N1)+(N2LOG2(N2)))".
229* 05 FILLER PIC X(14) VALUE IS SPACES.
230* 05 OUTPUT-VALUE-8 PIC Z(19).99.

```

NCR-IMDS CUBUL COMPILATION VERSION 05.12.00 01/03/14 10:10:35 51-MALSTEADS 00-UMALSTEADS

LINE DEBUG SEQ/NO A.....B.....C.....D.....E.....F.....G.....H.....I.....J.....K.....L.....M.....N.....O.....P.....Q.....R.....S.....T.....U.....V.....W.....X.....Y.....Z.....10.....

```

2300 / 01 PRINT-LINE-9. 2
2310 05 FILLER PIC X(28) VALUE IS SPACES. 2
2320 05 FILLER PIC X(14) 2
2330 05 FILLER VALUE IS "PROGRAM VOLUME". 2
2340 05 FILLER PIC X(21) VALUE IS SPACES. 2
2350 05 FILLER PIC X(22) 2
2360 05 FILLER VALUE IS "V = (21+2)LOG2(21+2)". 2
2370 05 FILLER PIC X(22) VALUE IS SPACES. 2
2380 05 OUTPUT-VALUE-9 PIC Z(9).9(4). 2
2390 2
2400 2
2410 01 PRINT-LINE-10. 2
2420 05 FILLER PIC X(28) VALUE IS SPACES. 2
2430 05 FILLER PIC X(20) 2
2440 05 FILLER VALUE IS "IMPLEMENTATION LEVEL". 2
2450 05 FILLER PIC X(15) VALUE IS SPACES. 2
2460 05 FILLER PIC X(33) 2
2470 05 FILLER VALUE IS "L = APPR. (2(22))/(21+2)". 2
2480 05 FILLER PIC X(11) VALUE IS SPACES. 2
2490 05 OUTPUT-VALUE-10 PIC Z(9).9(4). 2
2500 2
2510 01 PRINT-LINE-11. 2
2520 05 FILLER PIC X(28) VALUE IS SPACES. 2
2530 05 FILLER PIC X(16) 2
2540 05 FILLER VALUE IS "POTENTIAL VOLUME". 2
2550 05 FILLER PIC X(19) VALUE IS SPACES. 2
2560 05 FILLER PIC X(11) 2
2570 05 FILLER VALUE IS "V = (L+V)". 2
2580 05 FILLER PIC X(33) VALUE IS SPACES. 2
2590 05 OUTPUT-VALUE-11 PIC Z(9).9(4). 2
2600 2
2610 01 PRINT-LINE-12. 2
2620 05 FILLER PIC X(28) VALUE IS SPACES. 2
2630 05 FILLER PIC X(13) 2
2640 05 FILLER VALUE IS "EFFORT METRIC". 2
2650 05 FILLER PIC X(22) VALUE IS SPACES. 2
2660 05 FILLER PIC X(8) 2
2670 05 FILLER VALUE IS "EC = V/L". 2
2680 05 FILLER PIC X(36) VALUE IS SPACES. 2
2690 05 OUTPUT-VALUE-12 PIC Z(9).9(4). 2
2700 2
2710 01 PRINT-LINE-13. 2
2720 05 FILLER PIC X(28) VALUE IS SPACES. 2
2730 05 FILLER PIC X(16) 2
2740 05 FILLER VALUE IS "PROGRAMMING TIME". 2
2750 05 FILLER PIC X(19) VALUE IS SPACES. 2
2760 05 FILLER PIC X(9) 2
2770 05 FILLER VALUE IS "T = EC/18". 2
2780 05 FILLER PIC X(35) VALUE IS SPACES. 2
2790 05 OUTPUT-VALUE-13 PIC Z(9).9(4). 2
2800 05 FILLER PIC X(4) 2
2810 05 FILLER PIC X(4) 2
2820 05 FILLER VALUE IS "SEC.". 2

```

```

LINE  DEBUG SEQ/NO A...B..... ID.....
283* /
284* 01 PRINT-LINE-14.
285* 05 FILLER PIC X(107) VALUE IS SPACES.
286* 05 OUTPUT-VALUE-14 PIC Z(9)-9(4).
287* 05 FILLER PIC X VALUE IS SPACES.
288* 05 FILLER PIC X(4)
289* VALUE IS "MIN.".
290*
291* 01 PRINT-LINE-15.
292* 05 FILLER PIC X(107) VALUE IS SPACES.
293* 05 OUTPUT-VALUE-15 PIC Z(9)-9(4).
294* 05 FILLER PIC X VALUE IS SPACES.
295* 05 FILLER PIC X(4)
296* VALUE IS "HRS.".
297*
298* 01 PRINT-LINE-16.
299* 05 FILLER PIC X(128) VALUE IS SPACES.
300* 05 FILLER PIC X(14)
301* VALUE IS "LANGUAGE LEVEL".
302* 05 FILLER PIC X(21) VALUE IS SPACES.
303* 05 FILLER PIC X(9)
304* VALUE IS "ILLI(V)".
305* 05 FILLER PIC X(35) VALUE IS SPACES.
306* 05 OUTPUT-VALUE-16 PIC Z(9)-9(4).
307*
308* 01 PRINT-LINE-17.
309* 05 FILLER PIC X(25)
310* VALUE IS "FREQUENCY OF OPERATORS IN".
311* 05 FILLER PIC X(18)
312* VALUE IS " DESCENDING ORDER".
313*
314* 01 PRINT-LINE-18.
315* 05 OPERATOR-OUTPUT-COUNTER PIC Z(4).
316* 05 FILLER PIC X(5) VALUE IS SPACES.
317* 05 OPERATOR-OUTPUT-ELEMENT PIC X(30).
318*
319* 01 PRINT-LINE-19.
320* 05 FILLER PIC X(24)
321* VALUE IS "FREQUENCY OF OPERANDS IN".
322* 05 FILLER PIC X(18)
323* VALUE IS " DESCENDING ORDER".
324*
325* 01 PRINT-LINE-20.
326* 05 OPERAND-OUTPUT-COUNTER PIC Z(4).
327* 05 FILLER PIC X(4) VALUE IS SPACES.
328* 05 OPERAND-OUTPUT-ELEMENT PIC X(122).

```

```

LINE  DEBUG SEQ/NO A...B..... ID.....
3290  /
3300  77 THE-DATE PIC 9(16).
3310  77 PAGE-COUNT PIC 9(16).
3320  77 PTR PIC 9(18).
3330  77 START-OF-WORD PIC 9(18).
3340  77 END-OF-WORD PIC 9(18).
3350  77 WORD-SIZE PIC 9(18).
3360  77 SOURCE-PTR PIC 9(18).
3370  77 PTR-TEMP PIC 9(18).
3380  77 SUBSCRIPT-TEMP PIC 9(18).
3390  77 DELIMITER-TEMP PIC XX.
3400  77 SOURCE-WORD-TEMP PIC X(122).
3410  77 SOURCE-STOP PIC XX.
3420  77 SAVE-STOP PIC XX.
3430  77 ID-1 PIC 99.
3440  77 LINE-NUMBER PIC 9(4).
3450  77 LINE-NUMBER-DISPLAY PIC Z(4).
3460  77 OPERAND-KEY PIC 9(18).
3470  77 LINK-KEY PIC 9(18).
3480  77 NUMBER-OF-LINKS PIC 9(18).
3490  77 LAST-LINK PIC 9(18).
3500  77 LINK-INDEX PIC 9(18).
3510  77 ITEM-BEING-CONSIDERED PIC 9(18).
3520  77 ONE-TO-PRINT PIC 9(18).
3530  77 LAST-ONE-PRINTED PIC 9(18).
3540  77 CURRENT-HIGH PIC 9(18).
3550  77 LAST-HIGH PIC 9(18).
3560  77 NUMBER-OF-ITEMS-PRINTED PIC 9(18).

```

```

LINE  DEBUG SEQ/NO  A.....B.....ID.....
3570 /
3580 77 NUMBER-OF-UNIQUE-OPERANDS PIC 9(18) VALUE IS ZERO.
3590 77 NUMBER-OF-UNIQUE-OPERATORS PIC 9(18) VALUE IS ZERO.
3600 77 TOTAL-FREQUENCY-OF-OPERANDS PIC 9(18) VALUE IS ZERO.
3610 77 TOTAL-FREQUENCY-OF-OPERATORS PIC 9(18) VALUE IS ZERO.
3620 77 PROGRAM-VOCABULARY PIC 9(18) VALUE IS ZERO.
3630 77 PROGRAM-LENGTH PIC 9(18) VALUE IS ZERO.
3640 77 PROGRAM-VOLUME PIC 9(12)V9(4) VALUE IS ZERO.
3650 77 IMPLEMENTATION-LEVEL PIC 9(4)V9(4) VALUE IS ZERO.
3660 77 POTENTIAL-VOLUME PIC 9(14)V9(4) VALUE IS ZERO.
3670 77 EFFORT-METRIC PIC 9(14)V9(4) VALUE IS ZERO.
3680 77 ESTIMATE-OF-PROGRAM-LENGTH PIC 9(14)V9(4) VALUE IS ZERO.
3690 77 PROGRAMMING-TIME-SECONDS PIC 9(14)V9(4) VALUE IS ZERO.
3700 77 PROGRAMMING-TIME-MINUTES PIC 9(14)V9(4) VALUE IS ZERO.
3710 77 PROGRAMMING-TIME-HOURS PIC 9(14)V9(4) VALUE IS ZERO.
3720 77 LANGUAGE-LEVEL PIC 9(14)V9(4) VALUE IS ZERO.
3730 77 PASSED-TO-LOG-ROUTINE PIC 9(18) VALUE IS ZERO.
3740 77 LOG-BASE-TWO-RESULT PIC 9(3)V9(15) VALUE IS ZERO.
3750 77 TEMP-1 PIC 9(9)V9(9) VALUE IS ZERO.
3760 77 TEMP-2 PIC 9(18) VALUE IS ZERO.
3770 77 TEMP-3 PIC 9(18) VALUE IS ZERO.
3780 77 TEMP-4 PIC 9(18) VALUE IS ZERO.
3790 77 TEMP-5 PIC 9(18) VALUE IS ZERO.
3800 77 TEMP-6 PIC 9(18) VALUE IS ZERO.
3810 77 TEMP-7 PIC 9(4)V9(10) VALUE IS ZERO.
3820 77 HALF-LOG PIC 9(9)V9(9) VALUE IS ZERO.
3830 77 LOG-ADDEND PIC 9(3)V9(15) VALUE IS ZERO.
3840 77 LOG-E PIC 9(3)V9(15) VALUE IS ZERO.
3850 77 MULTIPLIER PIC 9(18) VALUE IS ZERO.
3860 77 FRACTION PIC 9(18) VALUE IS ZERO.
3870 77 I PIC 9(18) VALUE IS ZERO.
3880 77 K PIC 9(18) VALUE IS ZERO.
3890 77 X-PLUS-1 PIC 9(18) VALUE IS ZERO.
3900 77 X-MINUS-1 PIC 9(18) VALUE IS ZERO.
3910 77 CONVERSION-VALUE PIC 9(17) VALUE IS 1.4426950408893407.
3920

```


NCR-IMOS COBOL COMPILE VERSION 05-12-00 81/03/14 14:10:35 SIGNALSTEAD 80-DUALSTEAD

LINE DEBUG SEG/NO A...0.....: ID.....

```

3730 / 77 FILE-FLAG PIC 9 VALUE IS ZERO.
3740 88 FIXED-LENGTH-FILE VALUE IS ZERO.
3750 88 VARIABLE-LENGTH-FILE VALUE IS 1.
3760 77 EOF-FLAG PIC 9 VALUE IS ZERO.
3770 88 NOT-END-OF-FILE VALUE IS ZERO.
3780 88 END-OF-FILE VALUE IS 1.
3790 77 AREA-A-FLAG PIC 9 VALUE IS ZERO.
3800 88 AREA-A-OCCUPIED VALUE IS 1.
3810 77 CONSUME-SM PIC 9 VALUE IS ZERO.
3820 88 CONSUME-DONE VALUE IS 1.
3830 77 QUOTE-SM PIC 9 VALUE IS ZERO.
3840 88 BALANCED-QUOTE VALUE IS 1.
3850 77 DONE-SM PIC 9 VALUE IS ZERO.
3860 88 DONE VALUE IS 1.
3870 77 OPERAND-FIND-FLAG PIC 9 VALUE IS ZERO.
3880 88 OPERAND-NOT-FOUND VALUE IS ZERO.
3890 88 OPERAND-FOUND VALUE IS 1.
3900

```

```

LINE  DEBUG SEQ/NO  A...B..... ID.....
411*  / PROCEDURE DIVISION.
412*  *
413*  * DECLARATIVES.
414*  *
415*  *
416*  * OPEN-ERROR SECTION.
417*  * USE AFTER STANDARD EXCEPTION PROCEDURE ON FIXED-INPUT.
418*  *
419*  * OPEN-VARIABLE.
420*  * OPEN INPUT VARIABLE-INPUT.
421*  * MOVE 1 TO FILE-FLAG.
422*  * OPEN-END.
423*  *
424*  * END DECLARATIVES.

```

```

LINE  DEBUG SEQ/NO  A.....B.....C.....D.....E.....F.....G.....H.....I.....J.....K.....L.....M.....N.....O.....P.....Q.....R.....S.....T.....U.....V.....W.....X.....Y.....Z.....
4250  /
4260  /
4270  /
4280  /
4290  /
4300  /
4310  /
4320  /
4330  /
4340  /
4350  /
4360  /
4370  /
4380  /
4390  /
4400  /
4410  /
4420  /
4430  /
4440  /
4450  /
4460  /
4470  /
4480  /
4490  /
4500  /
4510  /
4520  /
4530  /
4540  /
4550  /
4560  /
4570  /
4580  /
4590  /
4600  /
4610  /
4620  /
4630  /
4640  /
4650  /
4660  /
4670  /
4680  /
4690  /
4700  /
4710  /
4720  /
4730  /
4740  /

LIST OF ROUTINES

0 COMMON SUBROUTINES
0.1 GET WORD
0.1.1 READ NEXT
0.1.2 ISOLATE WORD
0.1.2.1 CONSUME LITERAL
0.1.2.2 EXAMINE AND ISOLATE SUBSCRIPTS
0.2 GET TO TOP OF PAGE
0.3 PRINT BLANK LINE

1 MAIN ROUTINE

2 GET TIME AND DATE ROUTINE

3 INITIALIZE OPERANDR TABLE ROUTINE

4 INITIALIZE ALPHABET TABLE ROUTINE

5 PROCESS PROGRAM ID PARAGRAPH ROUTINE

6 ANALYZE PROCEDURE DIVISION ROUTINE
6.1 GET PAST AND REMOVE PARENS
6.2 REGISTER OPERATOR
6.3 REGISTER OPERAND

7 COMPUTE MEASURES ROUTINE
7.1 CALCULATE PROGRAM VOCABULARY
7.2 CALCULATE PROGRAM ELEMENTS
7.3 CALCULATE PROGRAM LENGTH
7.4 CALCULATE ESTIMATE OF LENGTH
7.4.1 COMPUTE LOG BASE TWO
7.5 CALCULATE IMPLEMENTATION LEVEL
7.6 CALCULATE POTENTIAL VOLUME
7.7 CALCULATE EFFORT METRIC
7.8 CALCULATE PROGRAMMING TIME
7.9 CALCULATE LANGUAGE LEVEL

8 OUTPUT RESULTS ROUTINE
8.1 OUTPUT MEASURED RESULTS
8.2 OUTPUT COMPUTED RESULTS
8.3 OUTPUT OPERATOR FREQUENCIES
8.4 OUTPUT OPERAND FREQUENCIES

```


LINE DEBUG SEG/ND A---B..... ID.....

```

5200 /
5210 .....
5220 .....
5230 .....
5240 .....
5250 .....
5260 00000000
5270 .....
5280 .....
5290 .....
5300 .....
5310 .....
5320 .....
5330 .....
5340 .....
5350 .....
5360 .....
5370 .....
5380 .....
5390 .....
5400 .....
5410 .....
5420 .....
5430 .....
5440 .....
5450 .....
5460 .....
5470 00000000
5480 00000000
5490 00000000
5500 .....
5510 .....
5520 .....
5530 .....
5540 00000000
5550 00000000
5560 .....
5570 .....
5580 .....
5590 .....
5600 00000000
5610 .....

```

0 COMMON SUBROUTINES

COMMON-SUBROUTINE SECTION.

0.1 GET WORD ROUTINE

FUNCTION: ISOLATE THE NEXT TOKEN IN THE COBOL SOURCE PROGRAM

INPUT: PTR

OUTPUT: SOURCE-WORD

LOCAL DATA: NONE

CALLED BY: PROCESS-PROGRAM-ID-PARAGRAPH

PROCESS-PROCEDURE-DIVISION

COLLECT-OPERATORS-AND-OPERANDS

GET-PAST-AND-REMOVE-PARENS

READ-NEXT

ISOLATE-WORD

CALLS:

GET-WORD.

PERFORM GET-WORD-ACTUAL.

IF NOT-END-OF-FILE

PERFORM GET-WORD-ACTUAL

UNTIL SOURCE-WORD NOT EQUAL SPACES

OR END-OF-FILE.

GET-WORD-ACTUAL.

IF PTR GREATER THAN 72

PERFORM READ-NEXT

PERFORM READ-NEXT

UNTIL NOT COMMENT-OR-DEBUG

OR END-OF-FILE.

IF NOT-END-OF-FILE

PERFORM ISOLATE-WORD.

LINE DEBUG SEQ/NO A...B..... ID.....

```

5620 /
5630 *
5640 *
5650 * O.1.1 READ NEXT ROUTINE
5660 *
5670 *
5680 *
5690 * FUNCTION: INPUTS THE NEXT CHARACTER OF THE COBOL
5700 * SOURCE FILE
5710 * INPUT: NONE
5720 * OUTPUT: PTR
5730 * SOURCE-LINE
5740 * LOCAL DATA: LINE-NUMBER
5750 * CALLED BY: GET-WORD-ACTUAL
5760 * CONSUME-LITERAL-CHARACTER
5770 * ABSORB-SPACES
5780 * CALLS: NONE
5790 *
5800 *
5810 *
5820 *
5830 *
5840 *
5850 *
5860 *
5870 *
5880 *
5890 *
5900 *
5910 *
5920 *
5930 *
5940 *
5950 *
5960 *
5970 *

```



```

LINE  DEBUG SEG/NO A..... ID.....
637* /
640*
641*
642* 0.1.2.1 CONSUME LITERAL ROUTINE
643*
644*
645*
646*
647*
648*
649*
650*
651*
652*
653*
654*
655*
656*
657*000144
658*000144
659*000148
660*00014C
661*000150
662*
663*00015C
664*
665*
666*000170
667*000170
668*000172
669*
670*
671*
672*
673*000192
674*
675*
676*
677*
678*
679*
680*00018C
681*00018C
682*0001CC
683*
684*0001DA
685*
686*0001DE
687*0001DE
688*
689*
690*
691*0001FC
692*
693*000200
694*000200

```

 CONSUME-LITERAL.
 MOVE ZERO TO SOURCE-PTR.
 MOVE ZERO TO CONSUME-SW.
 MOVE 1 TO QUOTE-SW.
 PERFORM CONSUME-LITERAL-CHARACTER
 UNTIL CONSUME-QUOTE.
 PERFORM ABSORB-SPACES
 UNTIL SOURCE-LINE-ITEM (PTR) NOT = SPACES.

 CONSUME-LITERAL-CHARACTER.
 ADD 1 TO SOURCE-PTR.
 IF PTR GREATER THAN 72
 PERFORM READ-NEXT
 PERFORM READ-NEXT
 UNTIL NOT COMMENT-OR-DEBUG
 PERFORM FLIP-QUOTE-SW.
 IF BALANCED-QUOTE
 IF SOURCE-LINE-ITEM (PTR) NOT EQUAL ""
 MOVE 1 TO CONSUME-SW
 MOVE SOURCE-LINE-ITEM (PTR) TO SOURCE-STOP
 ELSE PERFORM EAT-LITERAL-CHARACTER
 ELSE PERFORM EAT-LITERAL-CHARACTER.

 EAT-LITERAL-CHARACTER.
 MOVE SOURCE-LINE-ITEM (PTR) TO SOURCE-WORD-ITEM (SOURCE-PTR).
 IF SOURCE-LINE-ITEM (PTR) = ""
 PERFORM FLIP-QUOTE-SW.
 ADD 1 TO PTR.

 ABSORB-SPACES.
 IF PTR GREATER THAN 72
 PERFORM READ-NEXT
 PERFORM READ-NEXT
 UNTIL NOT COMMENT-OR-DEBUG.
 ADD 1 TO PTR.

 FLIP-QUOTE-SW.
 IF BALANCED-QUOTE

NCR-INDS CDBDL COMPILATION VERSION 05.12.00 81/03/14 18:10:35 SI-HALSTEAD5 BD-QHALSTEAD5

LINE DEBUG SEQ/NO A----- ID-----

4950 MOVE ZERO TO QUOTE-SW
 4960 ELSE MOVE 1 TO QUOTE-SW.

LINE DEBUG SEQ/NO A..... ID.....

```

697* /
698*
699*
700*
701*
702*
703*
704*
705*
706*
707*
708*
709*
710*
711*
712*
713*
714*
715*
716*
717*
718*000212
719*000212
720*
721*000218
722*00021E
723*
724*
725*
726*
727*
728*
729*
730*
731*
732*
733*
734*
735*
736*
737*000276
738*000276
739*000280
740*000286

*****
0.1.2.2 EXAMINE AND ISOLATE SUBSCRIPTS ROUTINE
*****
FUNCTIONS:
INPUT:
    STRIP OFF AND EXAMINE SUBSCRIPTS
    WORD-SIZE
    START-OF-WORD
    SOURCE-WORD
    SOURCE-WORD
    WORD-SIZE
    START-OF-WORD
    END-OF-WORD
    SOURCE-WORD-TEMP
    DELIMITER-TEMP
    PTR-TEMP
    ISOLATE-WORD
    NONE
OUTPUT:
LOCAL DATA:
    START-OF-WORD
    END-OF-WORD
    SOURCE-WORD-TEMP
    DELIMITER-TEMP
    PTR-TEMP
    ISOLATE-WORD
    NONE
CALLED BY:
CALLS:
*****
EXAMINE-AND-ISOLATE-SUBSCRIPTS.
ADD WORD-SIZE, START-OF-WORD
    GIVING END-OF-WORD.
SUBTRACT 1 FROM END-OF-WORD.
IF SOURCE-LINE-ITEM (END-OF-WORD) = " "
    AND SOURCE-WORD-ITEM (1) NOT EQUAL " ("
    AND SOURCE-WORD NOT EQUAL " )"
    MOVE 1 TO PTR-TEMP
    UNSTRING SOURCE-WORD
        DELIMITED BY " (" OR SPACES
    INFO SOURCE-WORD-TEMP
    DELIMITER IN DELIMITER-TEMP
    WITH POINTER PTR-TEMP
    IF DELIMITER-TEMP = " ("
        PERFORM BACK-UP-TO-BEFORE-SUBSCRIPT
    UNTIL SOURCE-LINE-ITEM (END-OF-WORD) = " ("
    MOVE END-OF-WORD TO PTR
    PERFORM BACK-UP-TO-BEFORE-SUBSCRIPT.
*****
BACK-UP-TO-BEFORE-SUBSCRIPT.
MOVE SPACES TO SOURCE-WORD-ITEM (WORD-SIZE).
SUBTRACT 1 FROM WORD-SIZE.
SUBTRACT 1 FROM END-OF-WORD.

```


NCR-INGOS COBOL COMPILATION VERSION 05.12.00 01/03/14 10:10:35 SI-HALSTEAD BU-QHALSTEAD

```

LINE  DEBUG SEQ/NO A..... ID.....
755*  /
756*  .....
767*  .....
768*  .....
769*  .....
770*  .....
771*  .....
772*  .....
773*  .....
774*  .....
775*  .....
776*  .....
777*  .....
778*  .....
779*  .....
780*00029E
781*00029E
782*000242

0.3 PRINT BLANK LINE ROUTINE

FUNCTION:  OUTPUT A BLANK LINE TO THE OUTPUT LISTING
INPUT:    NONE
OUTPUT:   PRINT-RECORD
LOCAL DATA:  NONE
CALLED BY:  OUTPUT-AN-OPERATOR
CALLS:     NONE

PRINT-BLANK-LINE.
MOVE SPACES TO PRINT-RECORD.
WRITE PRINT-RECORD AFTER ADVANCING 1 LINES.

```

```

LINE  DEBUG SEW/NO A..... ID.....
783*  / ..... 1
784*  ..... 01
785*  ..... 01
786*  ..... 01
787*  ..... 01
788*  ..... 01
789*  ..... 01
790*  ..... 01
791*  ..... 01
792*  ..... 01
793*  ..... 01
794*  ..... 01
795*  ..... 01
796*  ..... 01
797*  ..... 01
798*  ..... 01
799*  ..... 01
800*  ..... 01
801* 01020A ..... 1
802*  ..... 1
803* 01020A ..... 1
804* 01020A ..... 1
805* 01020 ..... 1
806* 01020E ..... 1
807* 0102EA ..... 1
808* 0102EE ..... 1
809* 0102F2 ..... 1

```

2 GET TIME AND DATE ROUTINE

FUNCTION: ACCEPT TIME AND DATE FOR LISTING
INPUT: TIME AND DATE FROM THE SYSTEM
OUTPUT: DATE-OUT
HOURS-OUT
MINUTES-OUT
SECONDS-OUT

LOCAL DATA: THE-TIME
THE-DATE
CALLED BY: MAIN
CALLS: NONE

TIME-AND-DATE SECTION 50.

GET-TIME-AND-DATE.
ACCEPT THE-TIME FROM TIME.
ACCEPT THE-DATE FROM DATE.
MOVE THE-DATE TO DATE-OUT.
MOVE HOURS TO HOURS-OUT.
MOVE MINUTES TO MINUTES-OUT.
MOVE SECONDS TO SECONDS-OUT.

```

LINE  DEBUG SEQ/NO A...:..... ID.....
810* /
811* .....
812* .....
813* .....
814* .....
815* .....
816* .....
817* .....
818* .....
819* .....
820* .....
821* .....
822* .....
823* .....
824* .....
825* .....
826* .....
827* .....
828* .....
829* .....
830* .....
831* .....
832* .....
833* .....
834* .....
835* .....
836* .....
837* .....
838* .....
839* .....
840* .....
841* .....
842* .....
843* .....
844* .....
845* .....
846* .....
847* .....
848* .....
849* .....
850* .....
851* .....
852* .....
853* .....
854* .....
855* .....
856* .....
857* .....
858* .....
859* .....
860* .....
861* .....
862* .....
863* .....
864* .....
865* .....

```

3 INITIALIZE OPERATOR TABLE ROUTINE

FUNCTION: INITIALIZE THE ENTRIES IN THE OPERATOR TABLE
INPUT: NONE
OUTPUT: OPERATOR-ENTRIES
LOCAL DATA: NONE
CALLED BY: MAIN
CALLS: NONE

INITIALIZE-TABLE SECTION 51.

INITIALIZE-OPERATOR-TABLE.
MOVE ZEROES TO OPERATOR-ENTRIES.
MOVE "ACCEPT"
TO OPERATOR-ELEMENT (1).
MOVE "ADD"
TO OPERATOR-ELEMENT (2).
MOVE "ADVANCING"
TO OPERATOR-ELEMENT (3).
MOVE "AFTER"
TO OPERATOR-ELEMENT (4).
MOVE "ALL"
TO OPERATOR-ELEMENT (5).
MOVE "ALTER"
TO OPERATOR-ELEMENT (6).
MOVE "AND"
TO OPERATOR-ELEMENT (7).
MOVE "AT"
TO OPERATOR-ELEMENT (8).
MOVE "BEFORE"
TO OPERATOR-ELEMENT (9).
MOVE "BY"
TO OPERATOR-ELEMENT (10).
MOVE "CALL"
TO OPERATOR-ELEMENT (11).
MOVE "CANCEL"
TO OPERATOR-ELEMENT (12).
MOVE "CLOSE"
TO OPERATOR-ELEMENT (13).
MOVE "COMPUTE"
TO OPERATOR-ELEMENT (14).
MOVE "CORR"
TO OPERATOR-ELEMENT (15).
MOVE "CORRESPONDING"
TO OPERATOR-ELEMENT (16).
MOVE "DELETE"
TO OPERATOR-ELEMENT (17).
MOVE "DELIMITED"
TO OPERATOR-ELEMENT (18).
MOVE "DELIMITER"
TO OPERATOR-ELEMENT (19).

```

LINE  UMBUC SEQ/NO  A.....B.....C.....D.....E.....F.....G.....H.....I.....J.....K.....L.....M.....N.....O.....P.....Q.....R.....S.....T.....U.....V.....W.....X.....Y.....Z.....
865*02034E      MOVE "DISABLE"
867*      TO OPERATOR-ELEMENT (20).
868*020354      MOVE "DISPLAY"
869*      TO OPERATOR-ELEMENT (21).
870*02035A      MOVE "DIVIDE"
871*      TO OPERATOR-ELEMENT (22).
872*020360      MOVE "DOWN"
873*      TO OPERATOR-ELEMENT (23).
874*020366      MOVE "ELSE"
875*      TO OPERATOR-ELEMENT (24).
876*02036C      MOVE "ENABLE"
877*      TO OPERATOR-ELEMENT (25).
878*020372      MOVE "END"
879*      TO OPERATOR-ELEMENT (26).
880*020378      MOVE "END-OF-PAGE"
881*      TO OPERATOR-ELEMENT (27).
882*02037E      MOVE "EQUAL"
883*      TO OPERATOR-ELEMENT (28).
884*020384      MOVE "GIVING"
885*      TO OPERATOR-ELEMENT (29).
886*02038A      MOVE "GO"
887*      TO OPERATOR-ELEMENT (30).
888*020390      MOVE "GREATER"
889*      TO OPERATOR-ELEMENT (31).
890*020396      MOVE "IF"
891*      TO OPERATOR-ELEMENT (32).
892*02039C      MOVE "IN"
893*      TO OPERATOR-ELEMENT (33).
894*0203A2      MOVE "INITIAL"
895*      TO OPERATOR-ELEMENT (34).
896*0203A8      MOVE "INSPECT"
897*      TO OPERATOR-ELEMENT (35).
898*0203AE      MOVE "INTO"
899*      TO OPERATOR-ELEMENT (36).
900*0203B5      MOVE "INVALID"
901*      TO OPERATOR-ELEMENT (37).
902*0203B0      MOVE "KEY"
903*      TO OPERATOR-ELEMENT (38).
904*0203C5      MOVE "LESS"
905*      TO OPERATOR-ELEMENT (39).
906*0203C0      MOVE "MESSAGE"
907*      TO OPERATOR-ELEMENT (40).
908*0203D5      MOVE "MOVE"
909*      TO OPERATOR-ELEMENT (41).
910*0203D0      MOVE "MULTIPLY"
911*      TO OPERATOR-ELEMENT (42).
912*0203E5      MOVE "NEXT"
913*      TO OPERATOR-ELEMENT (43).
914*0203ED      MOVE "NOT"
915*      TO OPERATOR-ELEMENT (44).
916*0203F5      MOVE "OF"
917*      TO OPERATOR-ELEMENT (45).
918*0203FD      MOVE "OPEN"
919*      TO OPERATOR-ELEMENT (46).
920*020405      MOVE "OR"
921*      TO OPERATOR-ELEMENT (47).

```

LINE DEBUG SEQ/NO A.....R..... ID.....

```

922*020400 MOVE "OUTPUT"
923* TO OPERATOR-ELEMENT (48).
924* MOVE "OVERFLOW"
925* TO OPERATOR-ELEMENT (49).
926* MOVE "PERFORM"
927* TO OPERATOR-ELEMENT (50).
928* MOVE "PROCEED"
929* TO OPERATOR-ELEMENT (51).
930* MOVE "READ"
931* TO OPERATOR-ELEMENT (52).
932* MOVE "RECEIVE"
933* TO OPERATOR-ELEMENT (53).
934* MOVE "RELEASE"
935* TO OPERATOR-ELEMENT (54).
936* MOVE "REPLACING"
937* TO OPERATOR-ELEMENT (55).
938* MOVE "RE-INDO"
939* TO OPERATOR-ELEMENT (56).
940* MOVE "RE-INIT"
941* TO OPERATOR-ELEMENT (57).
942* MOVE "ROUND"
943* TO OPERATOR-ELEMENT (58).
944* MOVE "SEARCH"
945* TO OPERATOR-ELEMENT (59).
946* MOVE "SEND"
947* TO OPERATOR-ELEMENT (60).
948* MOVE "SET"
949* TO OPERATOR-ELEMENT (61).
950* MOVE "SORT"
951* TO OPERATOR-ELEMENT (62).
952* MOVE "SORT-MERGE"
953* TO OPERATOR-ELEMENT (63).
954* MOVE "START"
955* TO OPERATOR-ELEMENT (64).
956* MOVE "STOP"
957* TO OPERATOR-ELEMENT (65).
958* MOVE "STRING"
959* TO OPERATOR-ELEMENT (66).
960* MOVE "SUBTRACT"
961* TO OPERATOR-ELEMENT (67).
962* MOVE "SUM"
963* TO OPERATOR-ELEMENT (68).
964* MOVE "TALLYING"
965* TO OPERATOR-ELEMENT (69).
966* MOVE "TERMINATE"
967* TO OPERATOR-ELEMENT (70).
968* MOVE "THAN"
969* TO OPERATOR-ELEMENT (71).
970* MOVE "TIMES"
971* TO OPERATOR-ELEMENT (72).
972* MOVE "TO"
973* TO OPERATOR-ELEMENT (73).
974* MOVE "UNSTRING"
975* TO OPERATOR-ELEMENT (74).
976* MOVE "UNTIL"
977* TO OPERATOR-ELEMENT (75).

```


LINE ULRJC SEQ/NO A.....B.....C.....D.....E.....F.....G.....H.....I.....J.....K.....L.....M.....N.....O.....P.....Q.....R.....S.....T.....U.....V.....W.....X.....Y.....Z.....

```

978*0204ED MOVE "UP" 1
979* TO OPERATOR-ELEMENT (76). 2
980*0204F5 MOVE "UPON" 3
981* TO OPERATOR-ELEMENT (77). 4
982*0204FD MOVE "VARYING" 5
983* TO OPERATOR-ELEMENT (78). 6
984*020505 MOVE "M" 7
985* TO OPERATOR-ELEMENT (79). 8
986*02050D MOVE "M" 9
987* TO OPERATOR-ELEMENT (80). 10
988*020515 MOVE "M" 11
989* TO OPERATOR-ELEMENT (81). 12
990*02051C MOVE "M" 13
991* TO OPERATOR-ELEMENT (82). 14
992*020523 MOVE "M" 15
993* TO OPERATOR-ELEMENT (83). 16
994*020528 MOVE "M" 17
995* TO OPERATOR-ELEMENT (84). 18
996*020533 MOVE "C" 19
997* TO OPERATOR-ELEMENT (85). 20
998*020538 MOVE "M" 21
999* TO OPERATOR-ELEMENT (86). 22
1000*020543 MOVE "M" 23
1001* TO OPERATOR-ELEMENT (87). 24
1002*02054A MOVE "M" 25
1003* TO OPERATOR-ELEMENT (88). 26
1004*020551 MOVE "CONVLT" 27
1005* TO OPERATOR-ELEMENT (89). 28
1006*020559 MOVE "ECHO" 29
1007* TO OPERATOR-ELEMENT (90). 30
1008*020561 MOVE "ERASE" 31
1009* TO OPERATOR-ELEMENT (91). 32
1010*020569 MOVE "ECHO" 33
1011* TO OPERATOR-ELEMENT (92). 34
1012*020571 MOVE "OVERLAP" 35
1013* TO OPERATOR-ELEMENT (93). 36
1014*020579 MOVE "PROMPT" 37
1015* TO OPERATOR-ELEMENT (94). 38
1016*020581 MOVE "SHIFT" 39
1017* TO OPERATOR-ELEMENT (95). 40
1018*020589 MOVE "TEST" 41
1019* TO OPERATOR-ELEMENT (96). 42
1020*020591 MOVE "TRANSPORT" 43
1021* TO OPERATOR-ELEMENT (97). 44
1022*020594 MOVE "WITH" 45
1023* TO OPERATOR-ELEMENT (98). 46

```


WCR-1M05 C80UL COMPILE VERSION 05.12.00 31/03/14 10:10:35 SI-HALSTEADS RO-HALSTEADS

LINE	DEBUG	SEQ/NO	A.....B.....		ID.....
1080	0206F3		MOVE 13 TO ALPHABET-LINK (13)•		3
1081	0206FC		MOVE 14 TO ALPHABET-LINK (14)•		3
1082	020705		MOVE 15 TO ALPHABET-LINK (15)•		4
1083	02070C		MOVE 16 TO ALPHABET-LINK (16)•		4
1084	020717		MOVE 17 TO ALPHABET-LINK (17)•		4
1085	02071F		MOVE 18 TO ALPHABET-LINK (18)•		3
1086	020728		MOVE 19 TO ALPHABET-LINK (19)•		3
1087	020731		MOVE 20 TO ALPHABET-LINK (20)•		3
1088	02073A		MOVE 21 TO ALPHABET-LINK (21)•		3
1089	020743		MOVE 22 TO ALPHABET-LINK (22)•		3
1090	02074C		MOVE 23 TO ALPHABET-LINK (23)•		3
1091	020755		MOVE 24 TO ALPHABET-LINK (24)•		3
1092	02075E		MOVE 25 TO ALPHABET-LINK (25)•		3
1093	020767		MOVE 26 TO ALPHABET-LINK (26)•		3

NCR-IMOS C080L COMPILATION VERSION 05.12.00 81/03/14 18:10:35 SI-HALSTEADS BU-DHALSTEADS

LINE ORBUG SEQ/NO A..... ID.....

```

11200 /
11270 *
11280 * 6 PROCESS PROCEDURE DIVISION ROUTINE
11300 *
11310 *
11320 * FUNCTION: SCAN THE SOURCE PROGRAM'S PROCEDURE DIVISION
11330 * AND COLLECT ALL OCCURRENCES OF OPERATORS
11340 * AND OPERANDS.
11350 * INPUT: SOURCE-WORD
11360 * SAVE-WORD
11370 * SOURCE-WORD
11380 * OUTPUT: DONE-SW
11390 * LOCAL DATA: OPERATOR-INDEX
11400 * CALLED BY: MAIN
11410 * CALLS: GET-PAST-AND-REMOVE-PARENS
11420 *
11430 *
11440 * ANALYZE-PROCEDURE-DIVISION SECTION 53.
11450 *
11460 * PROCESS-PROCEDURE-DIVISION.
11470 * PERFORM GET-WORD
11480 * UNTIL SOURCE-WORD = "PROCEDURE".
11490 * PERFORM GET-WORD.
11500 * PERFORM COLLECT-OPERATORS-AND-OPERANDS
11510 * UNTIL END-OF-FILE.
11520 *
11530 * COLLECT-OPERATORS-AND-OPERANDS.
11540 * PERFORM GET-WORD.
11550 * PERFORM GET-WORD
11560 * UNTIL END-OF-FILE
11570 * OR (SOURCE-WORD NOT = "SECTION"
11580 * AND NOT AREA-A-OCCUPIED).
11590 * IF NOT-END-OF-FILE
11600 * MOVE ZERO TO DONE-SW
11610 * PERFORM GET-PAST-AND-REMOVE-PARENS
11620 * UNTIL DONE
11630 * SET OPERATOR-INDEX TO 1.
11640 * SEARCH OPERATOR-TABLE VARYING OPERATOR-INDEX
11650 * AT END PERFORM REGISTER-OPERAND
11660 * WHEN OPERATOR-ELEMENT(OPERATOR-INDEX) = SAVE-WORD
11670 * PERFORM REGISTER-OPERATOR.

```


NCR-1405 COBOL COMPILATION VERSION 05.12.00 81/03/14 10:10:35 SI-HALSTEADS BD-DHALSTEADS

```

LINE  DEB06 SEQ/NO A.....: 10.....
/
12240 .....:
12250 .....:
12260 .....:
12270 .....:
12280 .....:
12290 .....:
12300 .....:
12310 .....:
12320 .....:
12330 .....:
12340 .....:
12350 .....:
12360 .....:
12370 .....:
12380 .....:
12390 .....:
12400 .....:
12410 .....:
12420 .....:
12430 .....:
12440 .....:
12450 .....:
12460 .....:
12470 .....:
12480 .....:
12490 .....:

* 6-2 REGISTER OPERATOR ROUTINE
*
* FUNCTION:  ENTER THE OPERATOR CURRENTLY BEING SCANNED
*            INTO THE OPERATOR TABLE BY INCREMENTING
*            THE COUNTER OF THE ENTRY ASSOCIATED WITH
*            THE PASSED INDEX
* INPUT:    TOTAL-FREQUENCY-OF-OPERATORS
*           OPERATOR-ENTRIES
*           OPERATOR-INDEX
* OUTPUT:   TOTAL-FREQUENCY-OF-OPERATORS
*           OPERATOR-ENTRIES
*           OPERATOR-INDEX
* LOCAL DATA:  NONE
* CALLED BY:   COLLECT-OPERATORS-AND-OPERANDS
* CALLS:       NONE
*
* REGISTER-OPERATOR.
* ADD 1 TO TOTAL-FREQUENCY-OF-OPERATORS.
* IF OPERATOR-COUNTER (OPERATOR-INDEX) = ZERO
*   ADD 1 TO NUMBER-OF-UNIQUE-OPERATORS.
* ADD 1 TO OPERATOR-COUNTER (OPERATOR-INDEX).

```

LINE DEBUG SEQ/NU A..... ID.....

```

12500 /
12510 *
12520 * 6-3 REGISTER OPERAND ROUTINE
12530 *
12540 *
12550 *
12560 * FUNCTION: REGISTER THE CURRENT OPERAND BEING SCANNED
12570 * BY PLACING IT IN THE OPERAND FILE IF IT
12580 * DOES NOT ALREADY HAVE AN ASSOCIATED ENTRY.
12590 * OR IF IT DOES, BY INCREMENTING THE COUNTER
12600 * OF THAT ENTRY. IN ORDER TO EXPEDITE THE
12610 * SEARCH FOR A GIVEN OPERAND A LINKED LIST
12620 * IS MAINTAINED FOR EACH LETTER OF THE
12630 * ALPHABET, PLUS A LINKED LIST OF ALL NON-
12640 * ALPHABETIC CHARACTERS.
12650 * INPUT: NUMBER-OF-UNIQUE-OPERANDS
12660 * TOTAL-FREQUENCY-OF-OPERANDS
12670 * SAVE-WORD
12680 * NUMBER-OF-LINKS
12690 * OPERAND FILE
12700 * LINK-FILE
12710 * NUMBER-OF-UNIQUE-OPERANDS
12720 * NUMBER-OF-LINKS
12730 * TOTAL-FREQUENCY-OF-OPERANDS
12740 * OPERAND-KEY
12750 * OPERAND-FIND-FLAG
12760 * OPERAND-COUNTER
12770 * LAST-LINK
12780 * LINK-KEY
12790 * CALLED BY: COLLECT-OPERATORS-AND-OPERANDS
12800 * CALLS: NONE
12810 *
12820 *
12830 * REGISTER-OPERAND.
12840 * ADD 1 TO TOTAL-FREQUENCY-OF-OPERANDS.
12850 * PERFORM DETERMINE-LINK-INDEX.
12860 * IF HEAD-LINK (LINK-INDEX) NOT EQUAL ZERO
12870 * MOVE HEAD-LINK (LINK-INDEX) TO LINK-KEY
12880 * MOVE ZERO TO OPERAND-FIND-FLAG
12890 * PERFORM GO-THRU-OPERANDS
12900 * UNTIL LINK-KEY = ZERO
12910 * OR OPERAND-FOUND
12920 * IF OPERAND-NOT-FOUND
12930 * ADD 1 TO NUMBER-OF-LINKS
12940 * MOVE LAST-LINK TO LINK-KEY
12950 * MOVE NUMBER-OF-LINKS TO FORWARD-LINK
12960 * PERFORM REWRITE-LINK-FILE
12970 * MOVE NUMBER-OF-LINKS TO LINK-KEY
12980 * MOVE ZERO TO FORWARD-LINK
12990 * ADD 1 TO NUMBER-OF-UNIQUE-OPERANDS
13000 * MOVE NUMBER-OF-UNIQUE-OPERANDS TO OPERAND-POINTER,
13010 * OPERAND-KEY
13020 * PERFORM WRITE-LINK-FILE
13030 * MOVE SAVE-WORD TO OPERAND-NAME
13040 * MOVE 1 TO OPERAND-COUNTER
13050 * PERFORM WRITE-OPERAND-FILE

```


MCRT-1M05 COBOL COMPILATION VERSION 05.12.00 81/03/14 18:10:35 ST-HALSTEAD BS-DHALSTEAD

```

LINE  DEBUG SEQ/NO 4..... ID.....
13620      DISPLAY "LINK KEY =", LINK-KEY      1
13630      STOP RUN.                             1
13640      *                                     1
13650      WRITE-OPERAND-FILE.                   1
13660      WRITE OPERAND-RECORD FROM OPERAND-ENTRY 1
13670      INVALID KEY                           1
13680      DISPLAY "RAN OUT OF SPACE IN OPERAND-FILE" 1
13690      DISPLAY "OPERAND KEY = ", OPERAND-KEY 1
13700      STOP RUN.                             1
13710      *                                     1
13720      REWRITE-OPERAND-FILE.                 1
13730      REWRITE OPERAND-RECORD FROM OPERAND-ENTRY 1
13740      INVALID KEY                           1
13750      DISPLAY "PROBLEM REWRITING OPERAND-FILE" 1
13760      DISPLAY "OPERAND KEY = ", OPERAND-KEY 1
13770      STOP RUN.                             1

```


NCM-1405 COBOL COMPILATION VERSION 05.12.00 01/03/14 1810135 SI-HALSTEADS RO-DHALSTEADS

LINE DEBUG SEQ/NO A...H.....: ID.....

```

14140 /
14150 .....: 1
14160 .....:
14170 * 7.1 CALCULATE PROGRAM VOCABULARY ROUTINE .....:
14180 .....:
14190 .....:
14200 * FUNCTION: COMPUTE THE PROGRAM VOCABULARY METRIC USING .....:
14210 * THE FORMULA: N1*N2 .....:
14220 * NUMBER-OF-UNIQUE-OPERATORS .....:
14230 * NUMBER-OF-UNIQUE-OPERANDS .....:
14240 * PROGRAM-VOCABULARY .....:
14250 * LOCAL DATA: NONE .....:
14260 * CALLED BY: COMPUTE-MEASURES .....:
14270 * CALLS: NONE .....:
14280 .....:
14290 .....:
14300 *0502EE CALCULATE-PROGRAM-VOCABULARY. .....:
14310 *0502EE ADD NUMBER-OF-UNIQUE-OPERATORS. .....:
14320 .....:
14330 * GIVING PROGRAM-VOCABULARY. .....:

```

NER-IMP05 COBOL COMPILE VERSION 05.12.00 01/03/14 10:10:35 SI-HALSTEAD RO-HALSTEAD

LINE DEBUG SEQ/NO A...B..... ID.....

```

1414 /
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453

```

7.2 CALCULATE PROGRAM LENGTH

FUNCTION: COMPUTE THE PROGRAM LENGTH METRIC USING THE
 FORMULA: F1+F2

INPUT: TOTAL-FREQUENCY-OF-OPERATORS
 TOTAL-FREQUENCY-OF-OPERANDS

OUTPUT: PROGRAM-LENGTH

LOCAL DATA: NONE

CALLED BY: COMPUTE-MEASURES

CALLS: NONE

CALCULATE-PROGRAM-LENGTH.
 ADD TOTAL-FREQUENCY-OF-OPERATORS,
 TOTAL-FREQUENCY-OF-OPERANDS
 GIVING PROGRAM-LENGTH.


```

LINE  DEBUG SEQ/NO  A...B..... ID.....
1508*  / .....
1509*  * .....
1510*  * 7.4.1 COMPUTE LOG BASE TWO ROUTINE .....
1511*  * .....
1512*  * .....
1513*  * .....
1514*  * .....
1515*  * .....
1516*  * FUNCTION: CALCULATE THE BASE TWO LOGARITHM OF THE .....
1517*  * INPUT: PASSED VALUE .....
1518*  * LOG-BASE-TWO-RESULT .....
1519*  * LOCAL DATA: HALF-LOG .....
1520*  * LOG-ADDEND .....
1521*  * X-PLUS-1 .....
1522*  * X-MINUS-1 .....
1523*  * TEMP-5 .....
1524*  * MULTIPLIER .....
1525*  * K .....
1526*  * I .....
1527*  * LOG-E .....
1528*  * FRACTION .....
1529*  * CALLED BY: CALCULATE-ESTIMATE-OF-LENGTH .....
1530*  * CALLS: CALCULATE-PROGRAM-VOLUME .....
1531*  * NONE .....
1532*  * .....
1533*  * COMPUTE-LOG-BASE-TWO. ....
1534*  * MOVE ZERO TO HALF-LOG. ....
1535*  * MOVE 1 TO LOG-ADDEND. ....
1536*  * ADD 1, PASSED-TO-LOG-ROUTINE .....
1537*  * GIVING X-PLUS-1. ....
1538*  * SUBTRACT 1 FROM PASSED-TO-LOG-ROUTINE .....
1539*  * GIVING X-MINUS-1. ....
1540*  * DIVIDE X-MINUS-1 BY X-PLUS-1 .....
1541*  * GIVING TEMP-5. ....
1542*  * MOVE TEMP-5 TO MULTIPLIER, HALF-LOG. ....
1543*  * MULTIPLY TEMP-5 BY TEMP-5 .....
1544*  * GIVING K. ....
1545*  * PERFORM COMPUTE-HALF-LOG .....
1546*  * VARYING I FROM 3 BY 2 .....
1547*  * UNTIL LOG-ADDEND = 0. ....
1548*  * MULTIPLY HALF-LOG BY 2 .....
1549*  * GIVING LOG-E. ....
1550*  * MULTIPLY LOG-E BY CONVERSION-VALUE .....
1551*  * GIVING LOG-BASE-TWO-RESULT ROUNDED. ....
1552*  * .....
1553*  * .....
1554*  * COMPUTE-HALF-LOG. ....
1555*  * DIVIDE 1 BY 1 .....
1556*  * GIVING FRACTION ROUNDED. ....
1557*  * MULTIPLY MULTIPLIER BY K .....
1558*  * GIVING MULTIPLIER. ....
1559*  * MULTIPLY MULTIPLIER BY FRACTION .....
1560*  * GIVING LOG-ADDEND ROUNDED. ....
1561*  * ADD LOG-ADDEND TO HALF-LOG. ....

```


MC2-INDS COBOL COMPILE VERSION 05.12.00 01/03/14 10:10:35 SI-MALSTEADS BU-MALSTEADS

LINE DENJG SEQ/NO A...B..... ID.....

```

1562* / .....
1563* .....
1564* .....
1565* .....
1566* .....
1567* .....
1568* .....
1569* .....
1570* .....
1571* .....
1572* .....
1573* .....
1574* .....
1575* .....
1576* .....
1577* .....
1578* .....
1579* .....
1580* .....
1581*050394 .....
1582*050394 .....
1583* .....
1584* .....
1585*050394 .....
1586* .....
1587* .....
1588*0503AD .....
1589* .....
1590* .....

/ .....
7.5 CALCULATE IMPLEMENTATION LEVEL ROUTINE .....
.....
FUNCTIONS COMPUTE THE IMPLEMENTATION LEVEL METRIC .....
USING THE FORMULA: .....
L = APPR. (ZEN11)/(IN21IF21) .....
INPUT: NUMBR-OF-UNIQUE-OPERANDS .....
OUTPUT: TOTAL-FREQUENCY-OF-OPERANDS .....
IMPLEMENTATION-LEVEL .....
TEMP-1 .....
TEMP-2 .....
LOCAL DATA: NONE .....
CALLED BY: COMPUTE-MEASURES .....
CALLS: NONE .....

CALCULATE-IMPLEMENTATION-LEVEL. ....
MULTIPLY NUMBER-OF-UNIQUE-OPERANDS .....
BY 2 .....
GIVING TEMP-2. ....
MULTIPLY NUMBER-OF-UNIQUE-OPERATORS .....
BY TOTAL-FREQUENCY-OF-OPERANDS .....
GIVING TEMP-3. ....
DIVIDE TEMP-2 .....
BY TEMP-3 .....
GIVING IMPLEMENTATION-LEVEL ROUNDED. ....

```

```

LINE  DEBUG SEQ/NO  A.....B.....: ID.....
15910 /
15920 *****
15930 * 7-5 CALCULATE POTENTIAL VOLUME ROUTINE *****
15940 *
15950 *
15960 *
15970 *
15980 *
15990 *
16000 *
16010 *
16020 *
16030 *
16040 *
16050 *
16060 *
16070 *
16080 *
16090 *
16100 *

FUNCTIONS  COMPUTE THE POTENTIAL VOLUME METRIC USING
            THE FORMULA:  $V_P = (L2IV)$ 
INPUT:     IMPLEMENTATION-LEVEL
            PROGRAM-VOLUME
OUTPUT:    POTENTIAL-VOLUME
            LOCAL DATA: NONE
            CALLED BY:  COMPUTE-MEASURES
            CALLS:     NONE

*****
CALCULATE-POTENTIAL-VOLUME.
MULTIPLY IMPLEMENTATION-LEVEL
BY PROGRAM-VOLUME
GIVING POTENTIAL-VOLUME.

```


NCR-IND'S CUDOL COMPILATION VERSION 05-12-00 81/03/14 10:10.35 SI-HALSTEAD'S RO-ONALSTEAD'S

LINE DEBUG SEQ/NO A...B..... ID.....

```

1635* /
1636* .....
1637* .....
1638* .....
1639* .....
1640* .....
1641* .....
1642* .....
1643* .....
1644* .....
1645* .....
1646* .....
1647* .....
1648* .....
1649* .....
1650* .....
1651* .....
1652* .....
1653* .....
1654* .....
1655* .....
1656* .....
1657* .....
1658* .....
1659* .....
1660* .....
1661* .....
1662* .....
1663* .....

/
7-8 CALCULATE PROGRAMMING TIME ROUTINE

FUNCTION: COMPUTE THE PROGRAMMING TIME METRIC IN TERMS
OF SECONDS, MINUTES, AND HOURS * USING
STROUD'S NUMBER OF 18 ELEMENTARY MENTAL
DISCRIMINATIONS PER SECOND

INPUT: EFFORT-METRIC
OUTPUT: PROGRAMMING-TIME-SECONDS
PROGRAMMING-TIME-MINUTES
PROGRAMMING-TIME-HOURS

LOCAL DATA: NONE
CALLED BY: CALCULATE-MEASURES
CALLS: NONE

CALCULATE-PROGRAMMING-TIME.
DIVIDE EFFORT-METRIC
BY 18
GIVING PROGRAMMING-TIME-SECONDS ROUNDED.
DIVIDE PROGRAMMING-TIME-SECONDS
BY 60
GIVING PROGRAMMING-TIME-MINUTES ROUNDED.
DIVIDE PROGRAMMING-TIME-MINUTES
BY 60
GIVING PROGRAMMING-TIME-HOURS ROUNDED.

```

MCR-1M05 CUBOL COMPILATION VERSION 05.12.00 01/03/14 10:10:33 SIGNALSTEADS NO-SIGNALSTEADS

LINE DEBUG SEQ/NO A...B..... ID.....

```

16640 / .....
16650 .....
16660 .....
16670 .....
16680 .....
16690 .....
16700 .....
16710 .....
16720 .....
16730 .....
16740 .....
16750 .....
16760 .....
16770 .....
16780 .....
16790 .....
16800 .....
16810 .....
16820 .....
16830 .....
16840 .....
16850 .....
16860 .....

      7.9 CALCULATE LANGUAGE LEVEL ROUTINE

      FUNCTION: COMPUTE THE LANGUAGE LEVEL METRIC USING THE
                  FORMULA: (L1/L1[V])
      INPUT: IMPLEMENTATION-LEVEL
              PROGRAM-VOLUME
              LANGUAGE-LEVEL
      OUTPUT: LOCAL DATA: TEMP-6
              CALLED BY: COMPUTE-MEASURES
              CALLS: NONE

      CALCULATE-LANGUAGE-LEVEL.
      MULTIPLY IMPLEMENTATION-LEVEL
      BY IMPLEMENTATION-LEVEL.
      GIVING TEMP-6.
      MULTIPLY TEMP-6
      BY PROGRAM-VOLUME
      GIVING LANGUAGE-LEVEL.

```



```

LINE  DEBUG SEQ/NO  A...B.....: ID.....
1716* /
1717* .....:
1718* .....:
1719* .....:
1720* .....:
1721* .....:
1722* .....:
1723* .....:
1724* .....:
1725* .....:
1726* .....:
1727* .....:
1728* .....:
1729* .....:
1730* .....:
1731* .....:
1732* .....:
1733* .....:
1734* .....:
1735* .....:
1736* .....:
1737* .....:
1738*0602EC .....:
1739*0602EC .....:
1740*0602FD .....:
1741* .....:
1742* .....:
1743*0602FA .....:
1744*0602FA .....:
1745*0602FE .....:
1746* .....:
1747* .....:
1748*060308 .....:
1749*060308 .....:
1750*06030C .....:
1751* .....:
1752* .....:
1753*060316 .....:
1754*060316 .....:
1755*06031A .....:
1756* .....:

      0.1 OUTPUT MEASURED RESULTS ROUTINE
      FUNCTION:  OUTPUT THE RESULTS OF THE MEASURED METRICS
                  TO THE PRINT-FILE
      INPUT:     NUMBER-OF-UNIQUE-OPERATORS
                  PRINT-LINE-2
                  NUMBER-OF-UNIQUE-OPERANDS
                  PRINT-LINE-3
                  TOTAL-FREQUENCY-OF-OPERATORS
                  PRINT-LINE-4
                  TOTAL-FREQUENCY-OF-OPERANDS
                  PRINT-LINE-5
      OUTPUT:    PRINT-FILE
      LOCAL DATA: NONE
      CALLED BY:  OUTPUT-RESULTS
      CALLS:      NONE

      OUTPUT-MEASURED-RESULT-1.
      MOVE NUMBER-OF-UNIQUE-OPERATORS TO OUTPUT-VALUE-2.
      WRITE PRINT-RECORD FROM PRINT-LINE-2
      AFTER ADVANCING 3 LINES.

      OUTPUT-MEASURED-RESULT-2.
      MOVE NUMBER-OF-UNIQUE-OPERANDS TO OUTPUT-VALUE-3.
      WRITE PRINT-RECORD FROM PRINT-LINE-3
      AFTER ADVANCING 2 LINES.

      OUTPUT-MEASURED-RESULT-3.
      MOVE TOTAL-FREQUENCY-OF-OPERATORS TO OUTPUT-VALUE-4.
      WRITE PRINT-RECORD FROM PRINT-LINE-4
      AFTER ADVANCING 2 LINES.

      OUTPUT-MEASURED-RESULT-4.
      MOVE TOTAL-FREQUENCY-OF-OPERANDS TO OUTPUT-VALUE-5.
      WRITE PRINT-RECORD FROM PRINT-LINE-5
      AFTER ADVANCING 2 LINES.

```

LINE DEBUG SEQ/NO A...B..... ID.....

```

1757* /
1758*
1759*
1760*
1761*
1762*
1763*
1764*
1765*
1766*
1767*
1768*
1769*
1770*
1771*
1772*
1773*
1774*
1775*
1776*
1777*
1778*
1779*
1780*
1781*
1782*
1783*
1784*
1785*
1786*
1787*
1788*
1789*
1790*
1791*
1792*
1793*060324
1794*060324
1795*060328
1796*
1797*
1798*060332
1799*060332
1800*060336
1801*
1802*
1803*060340
1804*060340
1805*060344
1806*
1807*
1808*06034E
1809*06034E
1810*060352
1811*
1812*

```

0.2 OUTPUT COMPUTED MEASURES ROUTINE

FUNCTION: OUTPUT THE RESULTS OF THE COMPUTED METRICS
TO THE PRINT-FILE

INPUT:
PROGRAM-VOCABULARY
PRINT-LINE-6
PROGRAM-LENGTH
PRINT-LINE-7
ESTIMATE-OF-PROGRAM-LENGTH
PRINT-LINE-8
PROGRAM-VOLUME
PRINT-LINE-9
IMPLEMENTATION-LEVEL
PRINT-LINE-10
POTENTIAL-VOLUME
PRINT-LINE-11
EFFORT-METRIC
PRINT-LINE-12
PROGRAMMING-TIME-SECONDS
PRINT-LINE-13
PROGRAMMING-TIME-MINUTES
PRINT-LINE-14
PROGRAMMING-TIME-HOURS
PRINT-LINE-15
LANGUAGE-LEVEL
PRINT-LINE-16
PRINT-FILE

OUTPUT:
LOCAL DATA: NONE
CALLED BY: OUTPUT-RESULTS
CALLS: NONE

OUTPUT-COMPUTED-RESULT-1.
MOVE PROGRAM-VOCABULARY TO OUTPUT-VALUE-6.
WRITE PRINT-RECORD FROM PRINT-LINE-6
AFTER ADVANCING 2 LINES.

OUTPUT-COMPUTED-RESULT-2.
MOVE PROGRAM-LENGTH TO OUTPUT-VALUE-7.
WRITE PRINT-RECORD FROM PRINT-LINE-7
AFTER ADVANCING 2 LINES.

OUTPUT-COMPUTED-RESULT-3.
MOVE ESTIMATE-OF-PROGRAM-LENGTH TO OUTPUT-VALUE-8.
WRITE PRINT-RECORD FROM PRINT-LINE-8
AFTER ADVANCING 2 LINES.

OUTPUT-COMPUTED-RESULT-4.
MOVE PROGRAM-VOLUME TO OUTPUT-VALUE-9.
WRITE PRINT-RECORD FROM PRINT-LINE-9
AFTER ADVANCING 2 LINES.

NCR-1140S CUBOL COMPILATION VERSION 05.12.00 01/03/14 1810:35 SI-HALSTEADS BO-DHALSTEADS

```

LINE  DEBUG SEQ/NO  A...B.....: ID.....
18430  / .....:
18440  * .....:
18450  * 0-3 OUTPUT OPERATOR FREQUENCIES ROUTINE .....:
18460  * .....:
18470  * .....:
18480  * .....:
18490  * .....:
18500  * FUNCTION:  OUTPUT THE OPERATORS FOUND IN THE SOURCE .....:
18510  *          PROGRAM IN ORDER OF DESCENDING FREQUENCY .....:
18520  *          TO THE PRINT-FILE .....:
18530  *          PRINT-LINE-17 .....:
18540  *          NUMBER-OF-UNIQUE-OPERATORS .....:
18550  *          PRINT-LINE-18 .....:
18560  *          OPERATOR-TABLE .....:
18570  *          PRINT-FILE .....:
18580  *          LOCAL DATA:  NUMBER-OF-ITEMS-PRINTED .....:
18590  *          LAST-ONE-PRINTED .....:
18600  *          LAST-HIGH .....:
18610  *          ITEM-BEING-CONSIDERED .....:
18620  *          ONE-TO-PRINT .....:
18630  *          CURRENT-HIGH .....:
18640  *          OPERATOR-OUTPUT-COUNTER .....:
18650  *          CALLED BY:  OUTPUT-RESULTS .....:
18660  *          CALLS:  GET-TO-TOP-OF-PAGE .....:
18670  *          PRINT-BLANK-LINE .....:
18680  * .....:
18690  * OUTPUT-OPERATOR-FREQUENCIES. .....:
18700  * PERFORM GET-TO-TOP-OF-PAGE. .....:
18710  * WRITE PRINT-RECORD FROM PRINT-LINE-17 .....:
18720  * AFTER ADVANCING 3 LINES. .....:
18730  * MOVE ZERO TO NUMBER-OF-ITEMS-PRINTED, .....:
18740  * LAST-ONE-PRINTED. .....:
18750  * MOVE 9999999999999999 TO LAST-HIGH. .....:
18760  * PERFORM OUTPUT-AN-OPERATOR .....:
18770  * UNTIL NUMBER-OF-ITEMS-PRINTED .....:
18780  *       = NUMBER-OF-UNIQUE-OPERATORS. .....:
18790  * .....:
18800  * OUTPUT-AN-OPERATOR. .....:
18810  * MOVE ZERO TO ITEM-BEING-CONSIDERED, .....:
18820  * CURRENT-HIGH, .....:
18830  * DONE-SM. .....:
18840  * PERFORM LOOK-FOR-HIGH-OPERATOR-COUNTER .....:
18850  * UNTIL ITEM-BEING-CONSIDERED = 98 .....:
18860  * OR DONE. .....:
18870  * MOVE ONE-TO-PRINT TO LAST-ONE-PRINTED. .....:
18880  * MOVE CURRENT-HIGH TO LAST-HIGH. .....:
18890  * MOVE OPERATOR-ELEMENT (ONE-TO-PRINT) TO .....:
18900  * OPERATOR-OUTPUT-ELEMENT. .....:
18910  * WRITE PRINT-RECORD FROM PRINT-LINE-18 .....:
18920  * AFTER ADVANCING 2 LINES .....:
18930  * AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE .....:
18940  * THRU PRINT-BLANK-LINE. .....:
18950  * .....:
18960  * ADD 1 TO NUMBER-OF-ITEMS-PRINTED. .....:
18970  * .....:
18980  * LOOK-FOR-HIGH-OPERATOR-COUNTER. .....:

```

NCR-IMUS COBOL COMPILE VERSION 05.12.00 01/03/14 14110135 ST=HALSTEAD BU=HALSTEAD

```

LINE  DEBUC SEQ/NO  A..... 10.....
1977*060430      ADD 1 TO ITEM-BEING-CONSIDERED.
1900*060433      IF OPERATOR-COUNTER(ITEM-BEING-CONSIDERED) > CURRENT-HIGH
1401*          AND OPERATOR-COUNTER(ITEM-BEING-CONSIDERED) < LAST-HIGH
1707*          MOVE ITEM-BEING-CONSIDERED TO ONE-TO-PRINT
1903*          MOVE OPERATOR-COUNTER(ITEM-BEING-CONSIDERED)
1904*              TO CURRENT-HIGH
1905*      ELSE IF OPERATOR-COUNTER(ITEM-BEING-CONSIDERED) = LAST-HIGH
1906*          AND ITEM-BEING-CONSIDERED > LAST-ONE-PRINTED
1907*          MOVE ITEM-BEING-CONSIDERED TO ONE-TO-PRINT
1908*          MOVE OPERATOR-COUNTER(ITEM-BEING-CONSIDERED)
1909*              TO CURRENT-HIGH
1910*          MOVE 1 TO DONE-SM.

```


MCR-1405 CORREL COMPILATION VERSION 05.17.00 01/03/14 10:10:35 SI=HALSTEAD5 UD=UNALSTEAD5

LINE DEBUC SEQ/NO A..... ID.....

```

1967*06051D      * ADD 1 TO NUMBER-OF-ITEMS-PRINTED.      :
1968*      :      :
1969*060522      LOOK-FOR-HIGH-OPERAND-COUNTER.      :
1970*060522      ADD 1 TO OPERAND-KEY.      :
1971*060524      READ OPERAND-FILE RECORD INTO OPERAND-ENTRY :
1972*      INVALID KEY      :
1973*      DISPLAY "OPERAND-FILE SEARCH ERROR"      :
1974*      STOP RUN.      :
1975*060537      IF OPERAND-COUNTER > CURRENT-HIGH      :
1976*      AND OPERAND-COUNTER < LAST-HIGH      :
1977*      MOVE OPERAND-KEY TO ONE-TO-PRINT      :
1978*      MOVE OPERAND-COUNTER TO CURRENT-HIGH      :
1979*      ELSE IF OPERAND-COUNTER = LAST-HIGH      :
1980*      AND OPERAND-KEY > LAST-ONE-PRINTED      :
1981*      MOVE OPERAND-KEY TO ONE-TO-PRINT      :
1982*      MOVE OPERAND-COUNTER TO CURRENT-HIGH      :
1983*      MOVE 1 TO DONE-SH.      :
1984*      ZZZZZZ END PROGRAM.      :

```

ADDRESS SIZE DEBUG ORDER TYPE

*** LOCAL ***

0000	480	ANS	0	FILE	FIXED-INPUT
0168	80	ANS	0	ALPHANUMERIC	INPUT-LINE
0048	512	ANS	0	FILE	VARIABLE-INPUT
0168	80	ANS	0	ALPHANUMERIC	INPUT-LINE
0090	132	ANS	0	FILE	PRINT-FILE
0188	132	ANS	0	ALPHANUMERIC	PRINT-RECORD
0008	420	GRP	0	FILE	OPERAND-FILE
023C	140	GRP	0	GROUP	OPERAND-RECORD
023C	122	ANS	0	ALPHANUMERIC	A-OPERAND-NAME
02B6	18	NSU	0	NUMERIC	OPERAND-REFERENCE-COUNTER
0120	504	GRP	0	FILE	LINK-FILE
02C8	36	GRP	0	GROUP	OPERAND-LINK-RECORD
02C8	14	NSU	0	NUMERIC	OPERAND-POINTER
02DA	18	NSU	0	NUMERIC	FORWARD-LINK
02FA	80	GRP	0	GROUP	SOURCE-LINE
02FA	6	ANS	0	ALPHANUMERIC	SEQUENCE-NUMBER
0300	1	ANS	0	ALPHANUMERIC	COMMENT-FIELD
0301	65	ANS	0	CONDITION-NAME	COMMENT-QR-DEBUG
0342	8	ANS	0	ALPHANUMERIC	LINE-CONTINUATION
02FA	80	GRP	1	GROUP	TEXT-LINE
02FA	1	GRP	1	ALPHANUMERIC	ID-FIELD
02FA	1	ANS	1	ALPHANUMERIC	SOURCE-LINE-REDEFINED
034A	122	ANS	0	ALPHANUMERIC	SOURCE-ITEMS
034A	122	GRP	0	GROUP	SOURCE-LINE-ITEM
034A	1	GRP	1	GROUP	SOURCE-WORD
034A	1	ANS	1	ALPHANUMERIC	SOURCE-WORD-REDEFINED
03C4	122	ANS	0	ALPHANUMERIC	SOURCE-WORDS
03C4	122	GRP	0	GROUP	SOURCE-WORD-ITEM
03C4	1	GRP	1	GROUP	SAVE-WORD
03C4	1	ANS	1	ALPHANUMERIC	SAVE-WORD-REDEFINED
043E	4704	GRP	0	GROUP	SAVE-WORDS
2694	4	GRP	0	INDEX-NAME	SAVE-WORD-ITEM
043E	48	GRP	1	GROUP	OPERATOR-ENTRIES
043E	30	ANS	1	ALPHANUMERIC	OPERATOR-INDEX
045C	18	NSU	1	NUMERIC	OPERATOR-TABLE
169E	78	GRP	0	GROUP	OPERATOR-ELEMENT
2698	4	GRP	0	INDEX-NAME	OPERATOR-COUNTER
169E	3	GRP	1	GROUP	ALPHABET-ENTRIES
169E	1	ANS	1	ALPHANUMERIC	ALPHABET-INDEX
					ALPHABET-TABLE
					ALPHABET-ITEM

ADDRESS	SIZE	DEBUG	UNDER	TYPE	NAME
169F	2	NSU	1	NUMERIC	ALPHABET-LINK
16EC	406	GRP	0	GROUP	OPERAND-HEAD-LINKS
16FC	18	GRP	1	GROUP	HEAD-LINK-LIST
16EC	14	NSU	1	NUMERIC	HEAD-LINK
1802	140	GRP	0	GROUP	OPERAND-ENTRY
1802	122	ANS	0	ALPHANUMERIC	OPERAND-NAME
194C	18	NSU	0	NUMERIC	OPERAND-COUNTER
195E	6	GRP	0	GROUP	THE-TIME
195E	2	NSU	0	NUMERIC	HOURS
1960	2	NSU	0	NUMERIC	MINUTES
1962	2	NSU	0	NUMERIC	SECONDS
1964	123	GRP	0	GROUP	HEADER
197A	30	ANS	0	ALPHANUMERIC	PROGRAM-NAME
1980	8	NSE	0	NUMERIC EDITED	DATE-OUT
19CA	2	NSU	0	NUMERIC	HOURS-OUT
19CD	2	NSU	0	NUMERIC	MINUTES-OUT
19D0	2	NSU	0	NUMERIC	SECONDS-OUT
19DC	3	ANS	0	ALPHANUMERIC	PAGE-NUMBER
19EO	119	GRP	0	GROUP	PRINT-LINE-1
1A58	116	GRP	0	GROUP	PRINT-LINE-2
1A8A	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-2
1ACC	116	GRP	0	GROUP	PRINT-LINE-3
1B2E	18	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-3
1B40	116	GRP	0	GROUP	PRINT-LINE-4
1B42	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-4
1B44	116	GRP	0	GROUP	PRINT-LINE-5
1C16	18	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-5
1C28	116	GRP	0	GROUP	PRINT-LINE-6
1C4A	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-6
1C9C	116	GRP	0	GROUP	PRINT-LINE-7
1CFC	18	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-7
1D10	119	GRP	0	GROUP	PRINT-LINE-8
1D78	12	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-8
1D98	121	GRP	0	GROUP	PRINT-LINE-9
1DF3	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-9
1E02	121	GRP	0	GROUP	PRINT-LINE-10
1E6D	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-10
1E7C	121	GRP	0	GROUP	PRINT-LINE-11
1EE7	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-11
1EF6	121	GRP	0	GROUP	PRINT-LINE-12

ADDRESS	SIZE	DEB	ORDER	TYPE	NAME
1F61	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-12
1F70	126	GRP	0	GROUP	PRINT-LINE-13
1F70	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-13
1FEE	126	GRP	0	GROUP	PRINT-LINE-14
2059	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-14
206C	126	GRP	0	GROUP	PRINT-LINE-15
20D7	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-15
20EA	121	GRP	0	GROUP	PRINT-LINE-16
2155	14	NSE	0	NUMERIC EDITED	OUTPUT-VALUE-16
2164	43	GRP	0	GROUP	PRINT-LINE-17
2190	39	GRP	0	GROUP	PRINT-LINE-18
2190	4	NSE	0	NUMERIC EDITED	OPERATOR-OUTPUT-COUNTER
2199	30	ANS	0	ALPHANUMERIC	OPERATOR-OUTPUT-ELEMENT
21B8	42	GRP	0	GROUP	PRINT-LINE-19
21E2	130	GRP	0	GROUP	PRINT-LINE-20
21E2	4	NSE	0	NUMERIC EDITED	OPERAND-OUTPUT-COUNTER
21EA	122	ANS	0	ALPHANUMERIC	OPERAND-OUTPUT-ELEMENT
2264	6	NSU	0	NUMERIC	THE-DATE
226A	3	NSU	0	NUMERIC	PAGE-COUNT
226D	18	NSU	0	NUMERIC	PTR
227F	18	NSU	0	NUMERIC	START-OF-WORD
2291	18	NSU	0	NUMERIC	END-OF-WORD
22A3	18	NSU	0	NUMERIC	WORD-SIZE
22B5	18	NSU	0	NUMERIC	SOURCE-PTR
22C7	18	NSU	0	NUMERIC	PTR-TEMP
22D9	18	NSU	0	NUMERIC	SUBSCRIPT-TEMP
22EB	2	ANS	0	ALPHANUMERIC	DELIMITER-TEMP
22ED	122	ANS	0	ALPHANUMERIC	SOURCE-WORD-TEMP
2367	2	ANS	0	ALPHANUMERIC	SOURCE-STOP
2369	2	ANS	0	ALPHANUMERIC	SAVE-STOP
236B	2	NSU	0	NUMERIC	ID-1
236D	4	NSU	0	NUMERIC	LINE-NUMBER

ADDRESS	SIZE	DEBUG	ORDER	TYPE	NAME
2371	4	NSF	0	NUMERIC EDITED	LINE-NUMBER-DISPLAY
2375	18	NSU	0	NUMERIC	OPERAND-KEY
2387	18	NSU	0	NUMERIC	LINK-KEY
2399	18	NSU	0	NUMERIC	NUMBER-OF-LINKS
23AB	18	NSU	0	NUMERIC	LAST-LINK
23BD	18	NSU	0	NUMERIC	LINK-INDEX
23CF	18	NSU	0	NUMERIC	ITEM-BEING-CONSIDERED
23E1	18	NSU	0	NUMERIC	ONE-TO-PRINT
23F3	18	NSU	0	NUMERIC	LAST-ONE-PRINTED
2405	18	NSU	0	NUMERIC	CURRENT-HIGH
2417	18	NSU	0	NUMERIC	LAST-HIGH
2429	18	NSU	0	NUMERIC	NUMBER-OF-ITEMS-PRINTED
243B	18	NSU	0	NUMERIC	NUMBER-OF-UNIQUE-OPERANDS
244D	18	NSU	0	NUMERIC	NUMBER-OF-UNIQUE-OPERATORS
245F	18	NSU	0	NUMERIC	TOTAL-FREQUENCY-OF-OPERANDS
2471	18	NSU	0	NUMERIC	TOTAL-FREQUENCY-OF-OPERATORS
2483	18	NSU	0	NUMERIC	PROGRAM-VOCABULARY
2495	18	NSU	0	NUMERIC	PROGRAM-LENGTH
24A7	18	NSU	0	NUMERIC	PROGRAM-VOLUME
24B7	6	NSU	0	NUMERIC	IMPLEMENTATION-LEVEL
24BD	18	NSU	0	NUMERIC	POTENTIAL-VOLUME
24CF	18	NSU	0	NUMERIC	EFFORT-METRIC
24E1	18	NSU	0	NUMERIC	ESTIMATE-OF-PROGRAM-LENGTH
24F3	18	NSU	0	NUMERIC	PROGRAMMING-TIME-SECONDS
2505	18	NSU	0	NUMERIC	PROGRAMMING-TIME-MINUTES
2517	18	NSU	0	NUMERIC	PROGRAMMING-TIME-HOURS
2529	18	NSU	0	NUMERIC	LANGUAGE-LEVEL
253B	18	NSU	0	NUMERIC	PASSED-TO-LOG-ROUTINE

ADDRESS	SIZE	DEBUG	ORDER	TYPE	NAME
254D	10	NSU	0	NUMERIC	LOG-BASE-TWO-RESULT
255F	10	NSU	0	NUMERIC	TEMP-1
2571	10	NSU	0	NUMERIC	TEMP-2
2583	10	NSU	0	NUMERIC	TEMP-3
2595	10	NSU	0	NUMERIC	TEMP-4
25A7	10	NSU	0	NUMERIC	TEMP-5
25B9	14	NSU	0	NUMERIC	TEMP-6
25C7	10	NSU	0	NUMERIC	TEMP-7
25D9	10	NSU	0	NUMERIC	HALF-LOG
25E0	10	NSU	0	NUMERIC	LOG-ADDEND
25FD	10	NSU	0	NUMERIC	LOG-E
260F	10	NSU	0	NUMERIC	MULTIPLIER
2621	10	NSU	0	NUMERIC	FRACTION
2633	10	NSU	0	NUMERIC	I
2645	10	NSU	0	NUMERIC	K
2657	10	NSU	0	NUMERIC	X-PLUS-1
2669	10	NSU	0	NUMERIC	X-MINUS-1
2670	10	NSU	0	NUMERIC	CONVERSION-VALUE
268D	1	NSU	0	NUMERIC	FILE-FLAG
				CONDITION-NAME	FIXED-LENGTH-FILE
				CONDITION-NAME	VARIABLE-LENGTH-FILE
268E	1	NSU	0	NUMERIC	EOF-FLAG
				CONDITION-NAME	NOT-END-OF-FILE
				CONDITION-NAME	END-OF-FILE
268F	1	NSU	0	NUMERIC	AREA-A-FLAG
				CONDITION-NAME	AREA-A-OCCUPIED
2690	1	NSU	0	NUMERIC	CONSUME-SM
				CONDITION-NAME	CONSUME-DONE
2691	1	NSU	0	NUMERIC	QUOTE-SM
				CONDITION-NAME	BALANCED-QUOTE
2692	1	NSU	0	NUMERIC	DONE-SM
				CONDITION-NAME	DONE

ADDRESS	SIZE	DEBUG	ORDER	TYPE	NAME
2693	1	NSU	0	NUMERIC CONDITION--NAME CONDITION--NAME	OPERAND-FIND-FLAG OPERAND-NOT-FOUND OPERAND-FOUND

REQUIRED RUNTIME ELEMENTS:

NUCLEUS
 ACCEPT/DISPLAY
 LINAGE
 MULTIPLY/DIVIDE
 STRING

LOCAL BYTE SIZE 12626
 GLOBAL BYTE SIZE 1150
 TOTAL BYTE SIZE 13704

NO ERRORS

NO WARNINGS

APPENDIX G

McCabe's Measure - Tables of Results

The following tables, in Figure G.1, Figure G.2, Figure G.3, Figure G.4, Figure G.5, and Figure G.6, present tabulations of the frequencies of modules associated with the given cyclomatic complexity values for the first and second versions of the three measured programs. The frequency of occurrence of the various complexity values is taken from the output listings that resulted from applying the tool to evaluate McCabe's cyclomatic complexity measure against the two versions of the programs.

<u>Cyclomatic Complexity</u>	<u>Number of Modules</u>
1	84
2	67
3	50
4	35
5	16
6	16
7	5
8	14
9	5
10	4
11	3
12	0
13	3
14	0
15	1
16	0
17	1
18	1
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

Figure G.1 Table of Cyclomatic Complexity Values Found in the First Version of Program A.

<u>Cyclomatic Complexity</u>	<u>Number of Modules</u>
1	96
2	111
3	64
4	33
5	20
6	29
7	11
8	11
9	7
10	5
11	4
12	3
13	2
14	0
15	1
16	1
17	1
18	1
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

Figure G.2 Table of Cyclomatic Complexity Values Found in the Second Version of Program A.

<u>Cyclomatic Complexity</u>	<u>Number of Modules</u>
1	73
2	59
3	44
4	27
5	22
6	7
7	3
8	4
9	6
10	1
11	0
12	4
13	2
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

Figure G.3 Table of Cyclomatic Complexity Values Found in the First Version of Program B.

<u>Cyclomatic Complexity</u>	<u>Number of Modules</u>
1	96
2	75
3	49
4	36
5	18
6	9
7	4
8	5
9	6
10	2
11	1
12	2
13	0
14	2
15	0
16	1
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

Figure G.4 Table of Cyclomatic Complexity Values Found in the Second Version of Program B.

<u>Cyclomatic Complexity</u>	<u>Number of Modules</u>
1	79
2	67
3	82
4	45
5	47
6	21
7	19
8	10
9	10
10	4
11	6
12	1
13	2
14	1
15	1
16	2
17	2
18	0
19	1
20	0
21	0
22	0
23	2
24	0
25	0
26	0
27	0
28	0

Figure G.5 Table of Cyclomatic Complexity Values Found in the First
Version of Program C.

<u>Cyclomatic Complexity</u>	<u>Number of Modules</u>
1	79
2	66
3	82
4	45
5	45
6	21
7	19
8	10
9	10
10	4
11	7
12	0
13	2
14	1
15	1
16	2
17	2
18	0
19	1
20	0
21	0
22	0
23	2
24	0
25	0
26	0
27	2
28	0

Figure G.6 Table of Cyclomatic Complexity Values Found in the Second Version of Program C.

APPENDIX H

Halstead's Metrics - Program A - Version 1

The following output listing is the partial results of applying the tool to evaluate Halstead's Software Science Metrics against the first version of Program A. For brevity, the ordered listing by frequency of operands is not shown here.

	NAME	DEFINITION	VALUE
MEASURED PROPERTIES:	NUMBER OF UNIQUE OPERATORS	N1	60
	NUMBER OF UNIQUE OPERANDS	N2	1013
	TOTAL FREQUENCY OF OPERATORS	F1	5702
	TOTAL FREQUENCY OF OPERANDS	F2	6220
COMPUTED MEASURES:	PROGRAM VOCABULARY	N1+N2	1073
	PROGRAM LENGTH	F1+F2	11922
	ESTIMATED PROGRAM LENGTH	$N^* = (N1)LOG2(N1) + (N2)LOG2(N2)$	10468.62
	PROGRAM VOLUME	$V = (F1+F2)LOG2(N1+N2)$	120223.9524
	IMPLEMENTATION LEVEL	$L = APPR. (2(N2))/((N1)(F2))$	-0054
	POTENTIAL VOLUME	$V* = (L)(V)$	648.1293
	EFFORT METRIC	EC = V/L	22109051.8435
	PROGRAMMING TIME	T = EC/10	2228280.6580 SEC. 20471.3443 MIN. 341.1491 HRS.
	LANGUAGE LEVEL	$((L)(L)(V))$	3.4998

FREQUENCY OF OPERATORS IN DESCENDING ORDER:

1131	TO
950	MOVE
628	-
606	PERFORM
480	IF
230	ELSE
172	NOT
149	DISPLAY
97	OR
92	UNTIL
90	OF
82	GO
82	(
72	ADD
54	SET
53	>
48	KEY
46	ACCEPT
44	BY
41	GIVING
38	PROMPT
36	SUBTRACT
35	INTO
33	NEXT
31	INVALID
29	AND
29	READ

27	AT
27	ERASE
26	END
19	COMPUTE
19	START
18	DELIMITED
17	UNSTRING
17	<
14	STRING
13	AFTER
13	UP
13	VARYING
12	ADVANCING
12	WITH
9	OPEN
8	CLOSE
5	DIVIDE
5	ECHO
4	INSPECT
4	STOP
4	EXOR
3	ALL
3	SEARCH
3	TALLYING
2	DELETE
2	DELIMITER
2	IN
2	MULTIPLY

1	DOH
1	OUTPUT
1	REPLACING
1	REWRITE
1	SHIFT

APPENDIX I

Halstead's Metrics - Program A - Version 2

The following output listing is the partial results of applying the tool to evaluate Halstead's Software Science Metrics against the second version of Program A. For brevity, the ordered listing by frequency of operands is not shown here.

	NAME	DEFINITION	VALUE
MEASURED PROPERTIES:	NUMBER OF UNIQUE OPERATORS	N1	61
	NUMBER OF UNIQUE OPERANDS	N2	1204
	TOTAL FREQUENCY OF OPERATORS	F1	7355
	TOTAL FREQUENCY OF OPERANDS	F2	8026
COMPUTED MEASURES:	PROGRAM VOCABULARY	N1+N2	1265
	PROGRAM LENGTH	F1+F2	15381
	ESTIMATED PROGRAM LENGTH	$N^* = [N1]LOG2[N1] + [N2]LOG2[N2]$	12683.05
	PROGRAM VOLUME	$V = [F1+F2]LOG2[N1+N2]$	158500.0001
	IMPLEMENTATION LEVEL	$L = APPR. [2(N2)]/[([N1])(F2)]$	.0049
	POTENTIAL VOLUME	$V^* = [L](V)$	776.6500
	EFFORT METRIC	EC = V/L	3222566.5814
	PROGRAMMING TIME	T = EC/18	1790314.2545 SEC. 29838.5704 MIN. 497.3095 HRS.
	LANGUAGE LEVEL	$[L][L](V)$	3.8055

FREQUENCY OF OPERATORS IN DESCENDING ORDER:

1435	TO
1221	MOVE
861	=
779	PERFORM
639	IF
340	ELSE
218	NOT
171	DISPLAY
134	1
132	OR
118	UNTIL
98	GO
94	ADD
88	OF
77	>
69	AND
61	ACCEPT
59	SET
53	BY
53	KEY
52	GIVING
50	PROMPT
49	SUBTRACT
37	ERASE
36	INTO
34	INVALID
33	NEXT

31	READ
27	AT
25	END
21	OPEN
20	CLOSE
20	COMPUTE
20	START
20	UNSTRING
20	<
19	AFTER
18	ADVANCING
17	DELIMITED
16	VARYING
14	UP
12	STRING
10	WITH
7	MULTIPLY
6	DIVIDE
5	STOP
4	INSPECT
4	OUTPUT
4	ECHO
4	EXOR
3	ALL
3	SEARCH
3	TALLYING
2	DELIMITER
2	IN

2	OVERFLOW
1	DELETE
1	DOWN
1	REPLACING
1	REWRITE
1	SHIFT

APPENDIX J

Halstead's Metrics - Program B - Version 1

The following output listing is the partial results of applying the tool to evaluate Halstead's Software Science Metrics against the first version of Program B. For brevity, the ordered listing by frequency of operands is not shown here.

	NAME	DEFINITION	VALUE
MEASURED PROPERTIES:	NUMBER OF UNIQUE OPERATORS	N1	47
	NUMBER OF UNIQUE OPERANDS	N2	744
	TOTAL FREQUENCY OF OPERATORS	F1	3595
	TOTAL FREQUENCY OF OPERANDS	F2	3760
COMPUTED MEASURES:	PROGRAM VOCABULARY	N1+N2	791
	PROGRAM LENGTH	F1+F2	7355
	ESTIMATED PROGRAM LENGTH	$N * = (N1)LOG2(N1) + (N2)LOG2(N2)$	7358.19
	PROGRAM VOLUME	$V = (F1+F2)LOG2(N1+N2)$	70810.5117
	IMPLEMENTATION LEVEL	$L = APPR. (2(N2)) / ((N1)(F2))$	.0084
	POTENTIAL VOLUME	$V * = (L)(V)$	594.8082
	EFFORT METRIC	EC = V/L	8409700.0181
	PROGRAMMING TIME	T = EC/10	467205.5566 SEC. 7786.7593 MIN. 129.7793 HRS.
	LANGUAGE LEVEL	$[L][L][V]$	4.9963

FREQUENCY OF OPERATORS IN DESCENDING ORDER:

580	TD
567	MOVE
439	PERFORM
418	"
331	IF
191	ELSE
89	NOT
79	!
77	UNTIL
70	OR
62	GO
53	DISPLAY
52	BY
51	>
44	ADD
43	SUBTRACT
32	VARYING
30	GIVING
24	AND
24	SET
23	INTO
19	ACCEPT
17	PROMPT
15	INVALID
15	OPEN
15	ERASE
13	READ

13	UNSTRING
13	UP
11	CLOSE
8	KEY
7	DELIMITED
7	NEXT
7	STRING
7	<
6	DIVIDE
5	COMPUTE
4	REWRITE
3	WITH
2	AFTER
2	MULTIPLY
2	OUTPUT
1	GREATER
1	STOP
1	THAN
1	EXOR
1	SHIFT

APPENDIX K

Halstead's Metrics - Program B - Version 2

The following output listing is the partial results of applying the tool to evaluate Halstead's Software Science Metrics against the second version of Program B. For brevity, the ordered listing by frequency of operands is not shown here.

	NAME	DEFINITION	VALUE
MEASURED PROPERTIES:	NUMBER OF UNIQUE OPERATORS	N1	48
	NUMBER OF UNIQUE OPERANDS	N2	845
	TOTAL FREQUENCY OF OPERATORS	F1	4430
	TOTAL FREQUENCY OF OPERANDS	F2	4670
COMPUTED MEASURES:	PROGRAM VOCABULARY	N1+N2	893
	PROGRAM LENGTH	F1+F2	9108
	ESTIMATED PROGRAM LENGTH	$N^0 = (N1 \log_2(N1) + N2 \log_2(N2))$	8483.85
	PROGRAM VOLUME	$V = (F1 + F2) \log_2(N1 + N2)$	89281.3190
	IMPLEMENTATION LEVEL	$L = \text{APPR. } [2(N2)] / [(N1)(F2)]$	.0075
	POTENTIAL VOLUME	$V^0 = (L)(V)$	669.6098
	EFFORT METRIC	EC = V/L	.11862476.0314
	PROGRAMMING TIME	T = EC/16	659026.4462 SEC. 10983.7741 MIN. 183.0629 HRS.
	LANGUAGE LEVEL	$(L)(L)(V)$	5.0220

FREQUENCY OF OPERATORS IN DESCENDING ORDER:

870	TO
602	MOVE
514	PERFORM
492	-
383	IF
219	ELSE
129	1
110	GO
102	NOT
78	UNTIL
74	OR
64	ADD
64	SUBTRACT
62	>
60	BY
56	DISPLAY
42	GIVING
37	OPEN
36	VARYING
34	INTO
32	AND
28	ACCEPT
24	SET
23	CLOSE
21	PROMPT
19	INVALID
17	NEXT

HALSTEAD'S COMPLEXITY MEASURE

PROGRAM-NAME = PROGRAM-8-VERSION-2

80/01/11

19:23:46

PAGE 003

16	READ
15	DIVIDE
15	UNSTRING
14	ERASE
13	UP
12	KEY
8	TIMES
7	DELIMITED
7	REWRITE
7	STRING
7	<
6	MULTIPLY
5	COMPUTE
3	AFTER
3	WITH
2	OUTPUT
2	STOP
1	GREATER
1	THAN
1	EXOR
1	SHIFT

APPENDIX L

Halstead's Metrics - Program C - Version 1

The following output listing is the partial results of applying the tool to evaluate Halstead's Software Science Metrics against the first version of Program C. For brevity, the ordered listing by frequency of operands is not shown here.

	NAME	DEFINITION	VALUE
MEASURED PROPERTIES:	NUMBER OF UNIQUE OPERATORS	N1	41
	NUMBER OF UNIQUE OPERANDS	N2	875
	TOTAL FREQUENCY OF OPERATORS	F1	9876
	TOTAL FREQUENCY OF OPERANDS	F2	12987
COMPUTED MEASURES:	PROGRAM VOCABULARY	N1+N2	916
	PROGRAM LENGTH	F1+F2	22863
	ESTIMATED PROGRAM LENGTH	$N^* = \{N1\} \log_2 \{N1\} + \{N2\} \log_2 \{N2\}$	8771.15
	PROGRAM VOLUME	$V = \{F1+F2\} \log_2 \{N1+N2\}$	224953.7162
	IMPLEMENTATION LEVEL	$L = \text{APPR. } \{F2\} / \{N1\} \{F2\}$	.0033
	POTENTIAL VOLUME	$V^* = \{L\} \{V\}$	742.3472
	EFFORT METRIC	EC = V/L	68445960.2303
	PROGRAMMING TIME	T = EC/18	3802553.3461 SEC. 63375.8891 MIN. 1056.2648 HRS.
	LANGUAGE LEVEL	$\{L\} \{L\} \{V\}$	2.4497

FREQUENCY OF OPERATORS IN DESCENDING ORDER:

1622	(
1501	TO
1433	MOVE
1118	-
778	PERFORM
689	IF
552	ELSE
354	OR
223	DISPLAY
154	ACCEPT
147	UNTIL
129	AFTER
124	ADVANCING
109	AND
105	NEXT
97	NOT
96	PROMPT
91	ADD
73	ECHO
63	>
51	GIVING
51	<
45	READ
39	SUBTRACT
30	DIVIDE
30	INTO
27	REWRITE

HALSTEAD'S COMPLEXITY MEASURE

25	ERASE
18	CLOSE
13	INVALID
13	KEY
12	STOP
11	BY
10	OPEN
10	VARYING
8	AT
8	END
6	ROUNDED
4	DELETE
4	OUTPUT
1	MULTIPLY

APPENDIX M

Halstead's Metrics - Program C - Version 2

The following output listing is the partial results of applying the tool to evaluate Halstead's Software Science Metrics against the second version of Program C. For brevity, the ordered listing by frequency of operands is not shown here.

MEASURED PROPERTIES:		NAME	DEFINITION	VALUE
		NUMBER OF UNIQUE OPERATORS	N1	41
		NUMBER OF UNIQUE OPERANDS	N2	870
		TOTAL FREQUENCY OF OPERATORS	F1	9942
		TOTAL FREQUENCY OF OPERANDS	F2	13200
COMPUTED MEASURES:		PROGRAM VOCABULARY	N1+N2	911
		PROGRAM LENGTH	F1+F2	23142
		ESTIMATED PROGRAM LENGTH	$N * = \{N1 * LOG2(N1) + N2 * LOG2(N2)\}$	8715.09
		PROGRAM VOLUME	$V = \{F1 * F2 * LOG2(N1 * N2)\}$	227516.1122
		IMPLEMENTATION LEVEL	$L = \text{APPR. } \{2 * IN2\} / \{N1\} * \{F2\}$	.0032
		POTENTIAL VOLUME	$V * = \{L\} * \{V\}$	728.0515
		EFFORT METRIC	EC = V/L	70765356.2770
		PROGRAMMING TIME	T = EC/16	3931408.6421 SEC. 65523.4780 MIN. 1092.0580 HRS.
		LANGUAGE LEVEL	$\{L\} * \{L\} * \{V\}$	2.3297

FREQUENCY OF OPERATORS IN DESCENDING ORDER:

1681	(
1508	TO
1437	MOVE
1115	=
794	PERFORM
686	IF
550	ELSE
352	OR
224	DISPLAY
152	ACCEPT
145	UNTIL
129	AFTER
124	ADVANCING
105	AND
105	NEXT
99	NOT
94	ADD
94	PROMPT
71	ECHO
61	>
51	GIVING
47	<
45	READ
41	SUBTRACT
30	DIVIDE
30	INTO
27	REWRITE

25	ERASE
18	CLOSE
13	INVALID
13	KEY
12	STOP
11	BY
10	OPEN
10	VARYING
8	AT
8	END
6	ROUNDED
4	DELETE
4	OUTPUT
1	MULTIPLY

APPENDIX N

Halstead's Metrics - Rank Order Frequency of Operators

The following graphs, in Figure N.1, Figure N.2, Figure N.3, Figure N.4, Figure N.5, and Figure N.6, present plots of the logarithmic values of the frequency of occurrence of operators in rank order. These plotted values correspond to the values for the frequency of operators found in the output listings in Appendix H through Appendix M, which are the partial results of applying the tool to evaluate Halstead's Software Science Metrics against the first and second versions of the three measured programs.

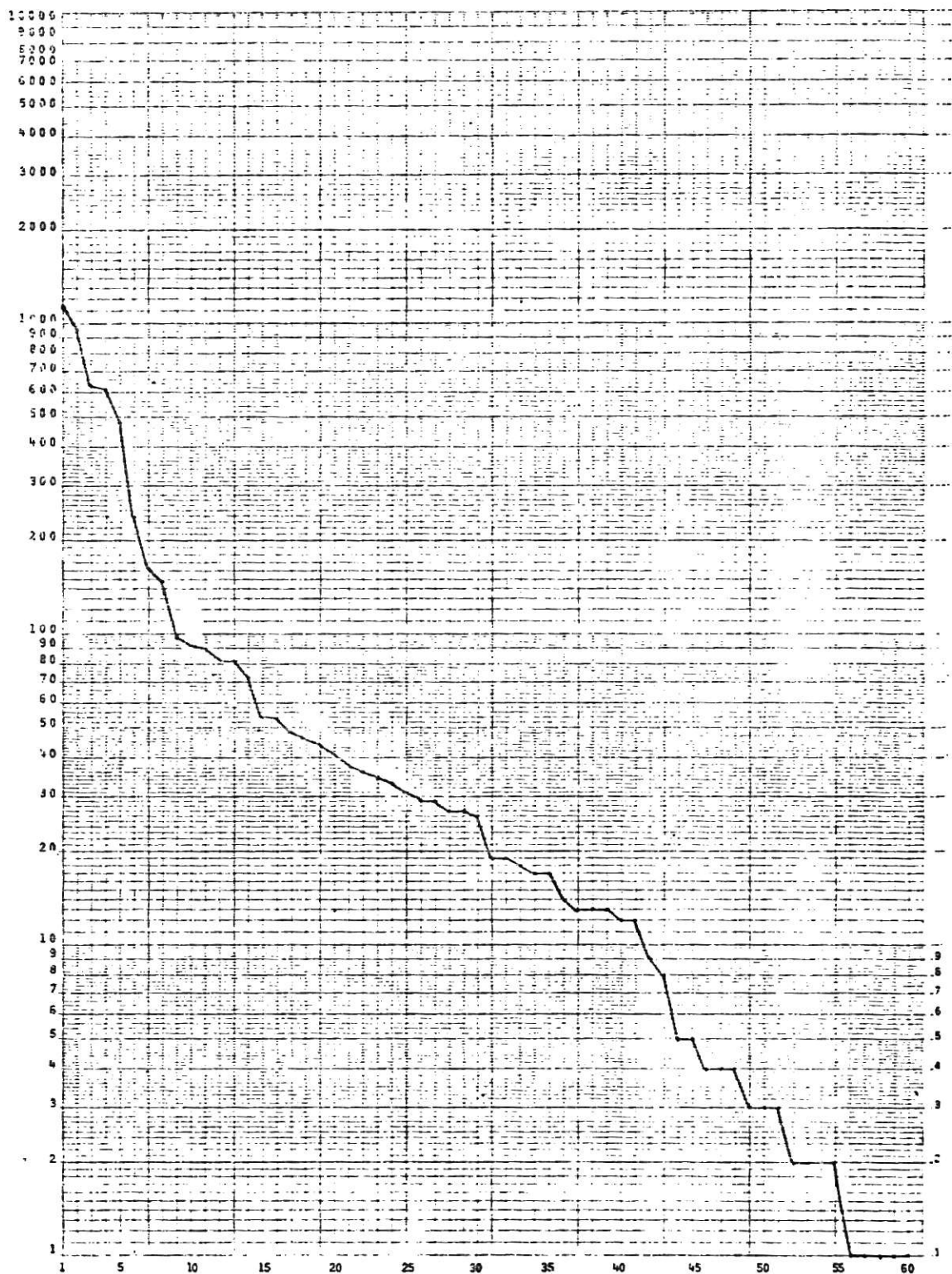


Figure N.1 Graph of Logarithmic Plot of Rank Order Frequency of Operators
for the First Version of Program A.

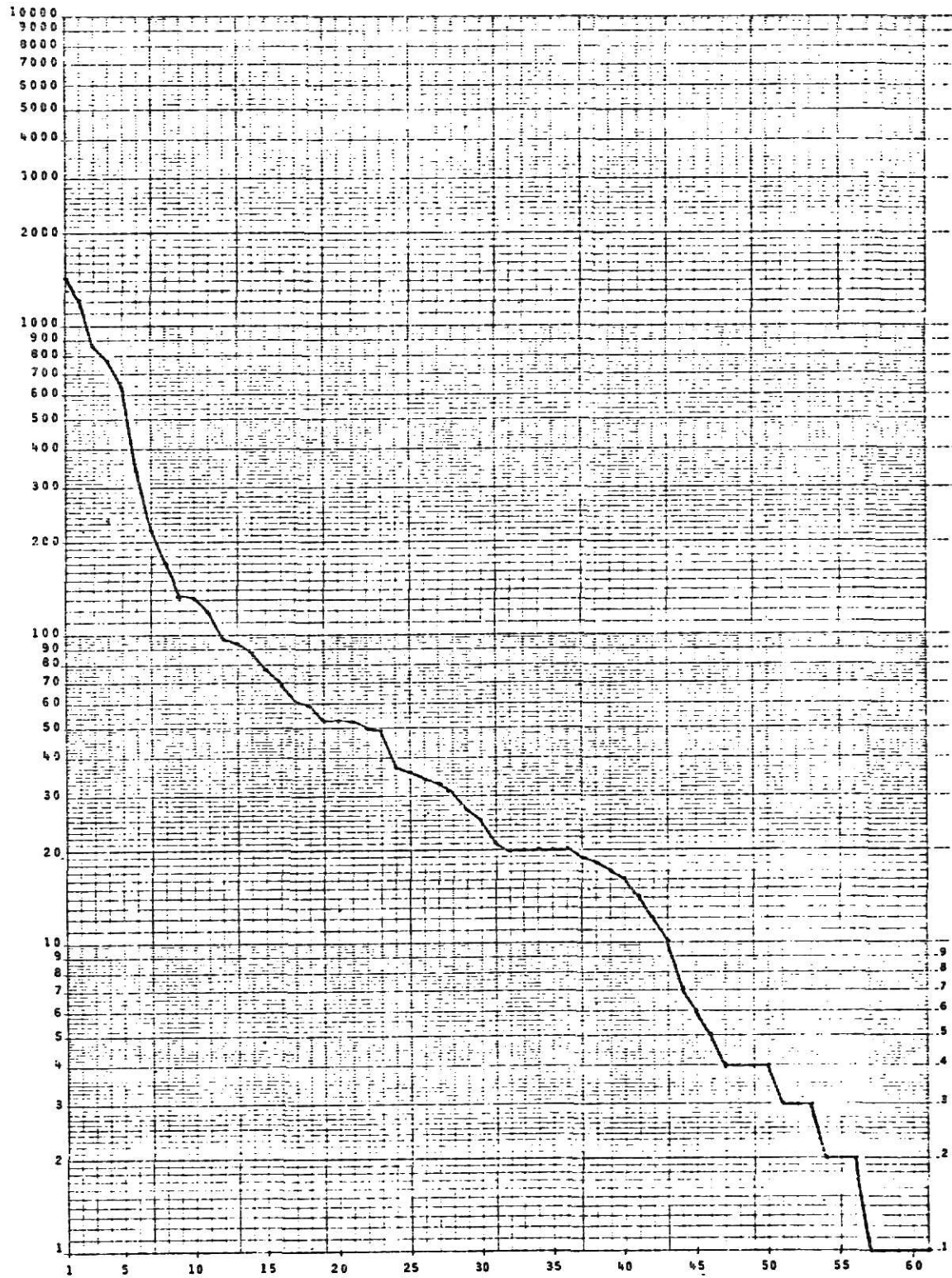


Figure N.2 Graph of Logarithmic Plot of Rank Order Frequency of Operators
for the Second Version of Program A.

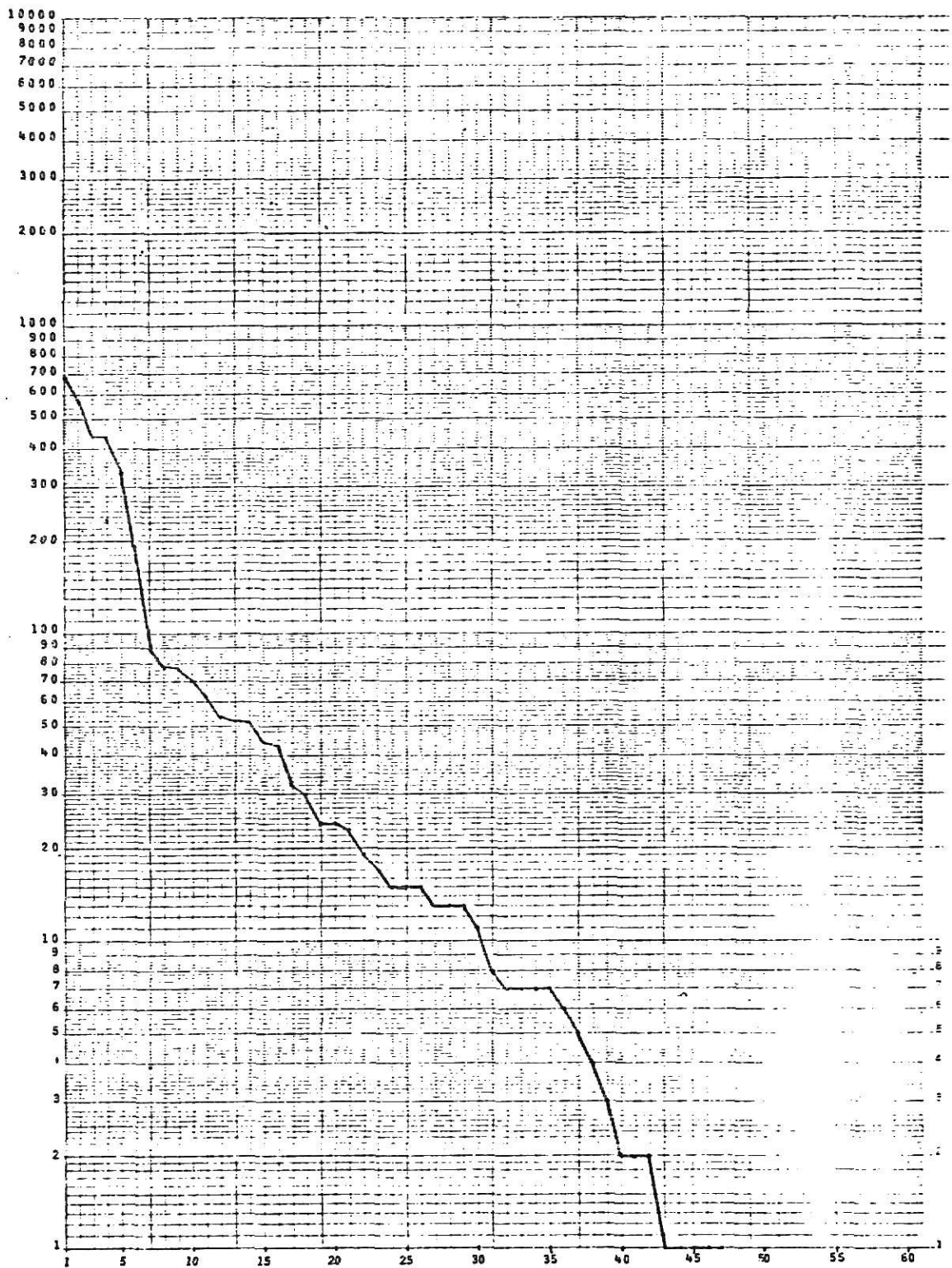


Figure N.3 Graph of Logarithmic Plot of Rank Order Frequency of Operators
for the First Version of Program B.

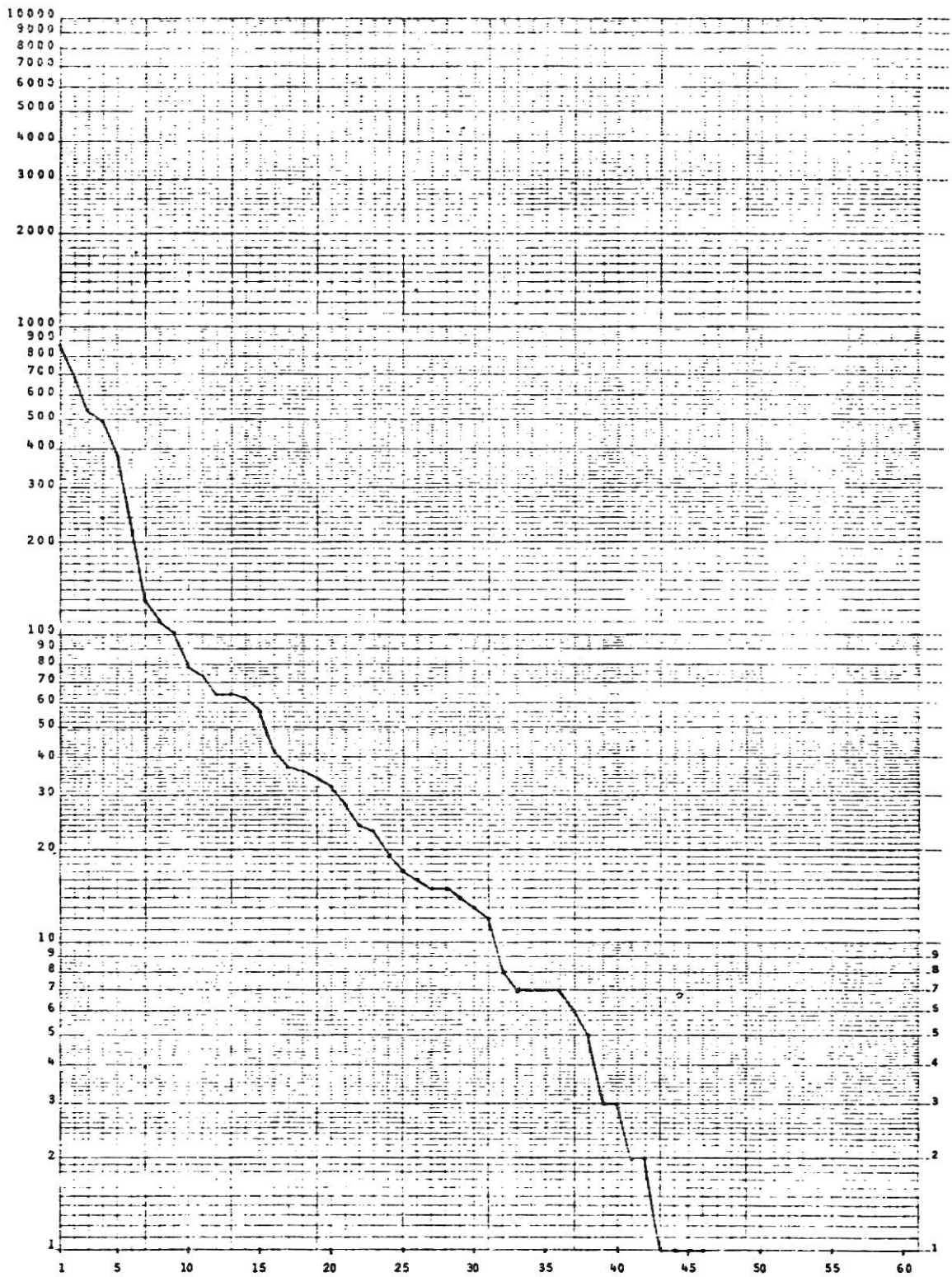


Figure N.4 Graph of Logarithmic Plot of Rank Order Frequency of Operators
for the Second Version of Program B.

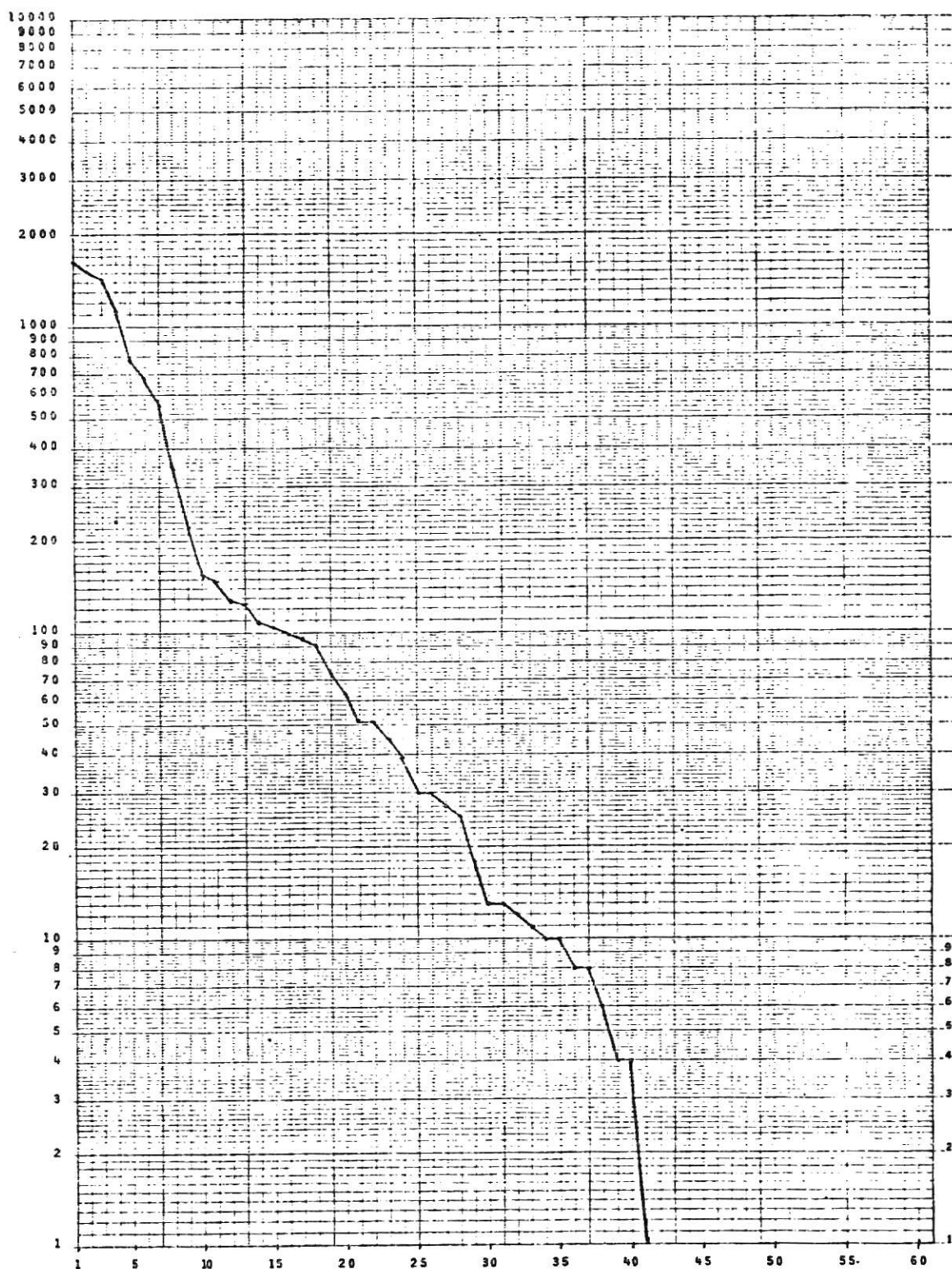


Figure N.5 Graph of Logarithmic Plot of Rank Order Frequency of Operators
for the First Version of Program C.

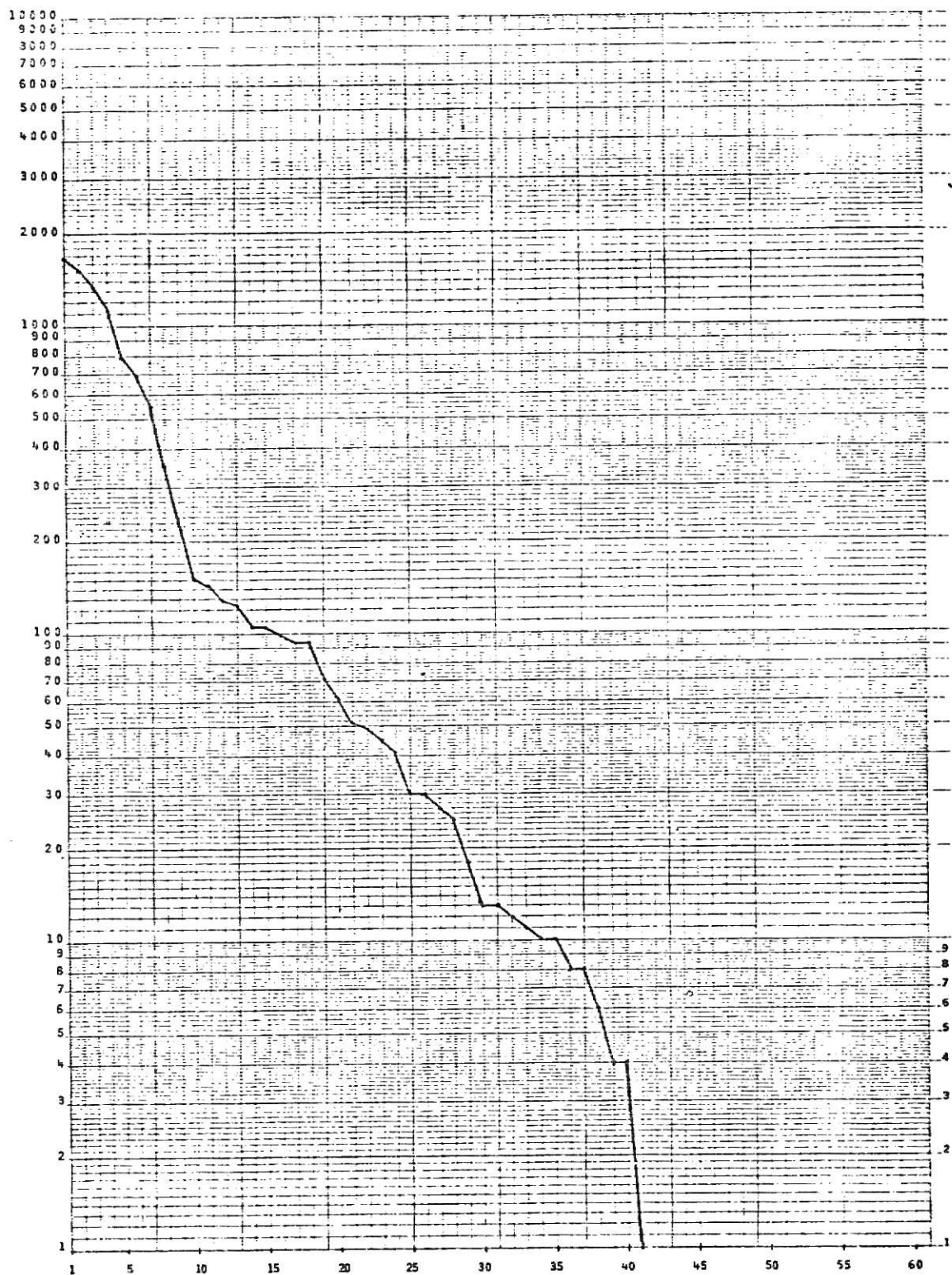


Figure N.6 Graph of Logarithmic Plot of Rank Order Frequency of Operators
for the Second Version of Program C.

AN EXPERIMENT IN THE IMPLEMENTATION AND
APPLICATION OF SOFTWARE COMPLEXITY MEASURES

by

RANDALL ROBERT MEALS

B. S., Iowa State University, 1977

AN 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

1981

Abstract

Much of the emphasis in adopting structured programming techniques is due to the perceived need to provide guidelines through which to reduce the complexity and increase the understandability of modules of software. A major factor motivating this emphasis is the high level of maintenance costs often experienced during the life-cycle of a piece of software. The complexity of a software module is somehow intuitively related to the difficulty experienced in developing, understanding, testing, modifying, and consequently maintaining that piece of software. With maintenance costs consuming the vast majority of the D. P. budgets of many organizations, there is a very real and pressing need to have some means by which to quantitatively measure software to determine if it conforms to programming style guidelines which enforce and reinforce adherence to whatever structured programming techniques an organization selects. Software complexity measures are an attempt to fill this need.

Many authors have presented what they view as a suitable means of measuring specific characteristics of software. After a survey of the literature available describing the various proposed complexity measures, those of M. Halstead's theory of Software Science and that of T. McCabe, his cyclomatic measure, were selected. A major reason for their selection was that they are relatively easy to automate, compared to many of the others.

The report serves as documentation of a project that had as its goal the selection of appropriate complexity measures, in this case those of Halstead and McCabe, and to automate and use the selected complexity measures to evaluate a set of large, commercial software. An important

aspect of this exercise was to determine if software which had a high complexity in terms of a specific complexity measure provided to be problematic in terms of error history.

The set of programs examined were not selected for any special reason, other than they were available, were sufficiently large, and had an associated error history. As is to be expected, the error history is not a complete record of all past and future errors in the selected software, but serves as a snapshot from which to detect tendencies and trends. The measured software is a set of three utility programs, written in COBOL, and have all been written by different programmers, in different circumstances, and have passed through the hands of several different individuals during the development, maintenance, and enhancement phases.

Examination of the results of applying McCabe's measure showed that the problem areas of code were not necessarily the paragraphs where the cyclomatic complexity exceeded a value of ten, but rather the problem areas were those where the average complexity was comparatively high. This is probably an indicator of the amount of mental effort required to understand the interaction between the affected paragraphs.

Examination of the results of applying Halstead's metrics showed a good relation between expected and actual error reports for two of the programs. The third program had the least error reports, although the highest number of errors was predicted. This third program also had the largest value for the Effort metric. The possible conclusion is that the third program is currently correct but hard to understand, and thus potentially will be hard to modify.

This project has demonstrated that it is reasonable and useful to automate the selected complexity measures via coding in COBOL to apply

against COBOL source programs. Although they certainly do not serve as complete indicators of complexity, the measures appear to serve as useful indicators of not only error-prone sections of code, by as warning flags for otherwise innocent appearing pieces of software. The author is proposing that in the future tools for the two measures for each programming language used be implemented and used as part of the standard development and maintenance process.