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:
# TABLE OF CONTENTS

**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 Compatibility 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
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 Compatibility 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
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Proposed Complexity Measures</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of Flow Graph where $V(G) = 4$</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Example of Flow Graph where $V(G) = 7$</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Example of Flow Graph where $V(G) = 5$</td>
<td>11</td>
</tr>
<tr>
<td>4.1</td>
<td>Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program A</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program B</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>Frequency of Occurrence of Cyclomatic Complexity Values for the First and Second Versions of Program C</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>Frequency of Occurrence of Cyclomatic Complexity Values for the First Versions of the Measured Programs</td>
<td>33</td>
</tr>
<tr>
<td>4.5</td>
<td>Frequency of Occurrence of Cyclomatic Complexity Values for the Second Versions of the Measured Programs</td>
<td>34</td>
</tr>
<tr>
<td>4.6</td>
<td>Summary of Results of Applying McCabe's Cyclomatic Complexity Measure</td>
<td>35</td>
</tr>
<tr>
<td>4.7</td>
<td>Comparison of Actual Error Count of Estimated Error Count</td>
<td>38</td>
</tr>
<tr>
<td>C.1</td>
<td>General Program Flow Through McCabe's Complexity Measure Evaluation Tool</td>
<td>C-5</td>
</tr>
<tr>
<td>D.1</td>
<td>General Program Flow Through Halstead's Metrics Evaluation Tool</td>
<td>D-7</td>
</tr>
<tr>
<td>G.1</td>
<td>Table of Cyclomatic Complexity Values Found in the First Version of Program A</td>
<td>G-1</td>
</tr>
<tr>
<td>G.2</td>
<td>Table of Cyclomatic Complexity Values Found in the Second Version of Program A</td>
<td>G-2</td>
</tr>
</tbody>
</table>
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 -
1.1 **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
<table>
<thead>
<tr>
<th>Orientation of Measure</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Flow</td>
<td>Cyclomatic Number</td>
<td>McCabe [1976]</td>
</tr>
<tr>
<td></td>
<td>Count of Program Paths</td>
<td>Sullivan [1973]</td>
</tr>
<tr>
<td></td>
<td>Enhancement of Cyclomatic Number (includes a count of logical conditions)</td>
<td>Meyers [1977]</td>
</tr>
<tr>
<td></td>
<td>Measurement by the Pair (Cyclomatic Number, Operator Count)</td>
<td>Hansen [1978]</td>
</tr>
<tr>
<td></td>
<td>Number of Multiple Entry Loops</td>
<td>Peterson [1973]</td>
</tr>
<tr>
<td></td>
<td>Number of Knots</td>
<td>Woodward, et al. [1979]</td>
</tr>
<tr>
<td></td>
<td>Cyclomatic Complexity Interval Plus # lines into/out of line of code</td>
<td>Cobb [1978]</td>
</tr>
<tr>
<td>Module Interaction</td>
<td>Number of Modules or Subsystems</td>
<td>Gilb [1977]</td>
</tr>
<tr>
<td></td>
<td>$R \left( \frac{\text{Number of Module linkages}}{\text{Number of Modules}} \right)$</td>
<td>Gilb [1977]</td>
</tr>
<tr>
<td></td>
<td>Measure based on control structures and control variables</td>
<td>McClure [1978]</td>
</tr>
<tr>
<td>Data Reference</td>
<td>Measure of difficulty in understanding software's function based on components of sets on input and output</td>
<td>Chaplin [1979]</td>
</tr>
<tr>
<td>Program Control</td>
<td>Minimal Intersection Number</td>
<td>Chen [1979]</td>
</tr>
<tr>
<td>Logical Complexity</td>
<td>$R \left( \frac{\text{Number of non-normal exits from a decision statement}}{\text{Total number of instructions}} \right)$</td>
<td>Gilb [1977]</td>
</tr>
<tr>
<td>Software Science</td>
<td>Metrics of software science predict complexity of a program</td>
<td>Halstead [1977]</td>
</tr>
<tr>
<td></td>
<td>Approach complexity via statistical (natural) language</td>
<td>Shoemaker &amp; Laemkei [1977]</td>
</tr>
<tr>
<td>Composite Measure of</td>
<td>Index of Complexity based on Structure/Interaction/Instruction Mix/Data Reference Program Characteristics</td>
<td>Zolnowski &amp; Simmons [1977]</td>
</tr>
<tr>
<td>Complexity</td>
<td>Interface complexity/Computational complexity/I/O complexity/Readability</td>
<td>Thayer [1976]</td>
</tr>
</tbody>
</table>

**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
\((p = 1 \text{ for single entry/single exit modules})\).

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
\( \pi \) = 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 complex-
ity. The essential complexity, \( e_v \), 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) \( n_1 \) - the number of unique operators,

2) \( n_2 \) - 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 - \( n = n_1 + n_2 \)

2) Program Length - \( N = N_1 + N_2 \)

3) Estimated Program Length - \( \hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2 \)

4) Program Volume - \( V = (N_1 + N_2)\log_2(n_1 + n_2) \)

5) Implementation Level - \( L = V^*/V \)

6) Estimated Implementation Level - \( \hat{L} = (2/n_1)(n_2/N_2) \)

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, 1B

10) Language Level - \( \lambda = (L)^2V \)
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, \( E_e \), 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, 1965], and who first reported this rate, which, for time in seconds, in taken as 18 [Halstead, 1979].

The equation for determining the Language Level, \( \lambda \), 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 accommodate 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 writ-
ten 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, espe-
cially the parsing of specific COBOL syntactic structures in the Pro-
cedure 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-quotiation 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-alphabets 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 throwaway 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 that 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.
<table>
<thead>
<tr>
<th>Version</th>
<th>Number of Problem Reports</th>
<th>Total # of Modules</th>
<th>Total # of Decision Points</th>
<th>Average Cyclomatic Complexity Value of Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program A</td>
<td>1</td>
<td>304</td>
<td>1037</td>
<td>3.4112</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>32</td>
<td>1362</td>
<td>3.5561</td>
</tr>
<tr>
<td>Program B</td>
<td>1</td>
<td>252</td>
<td>744</td>
<td>3.0714</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>904</td>
<td>2.9446</td>
</tr>
<tr>
<td>Program C</td>
<td>1</td>
<td>404</td>
<td>1662</td>
<td>4.1139</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>1648</td>
<td>4.1097</td>
</tr>
</tbody>
</table>

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].
<table>
<thead>
<tr>
<th>Program</th>
<th>Actual Error Count $\beta_v$</th>
<th>Estimated Error Count $\hat{\beta}_v$ for Version 1</th>
<th>Estimated Error Count $\hat{\beta}_v$ for Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program A</td>
<td>32</td>
<td>40.01</td>
<td>52.83</td>
</tr>
<tr>
<td>Program B</td>
<td>11</td>
<td>23.60</td>
<td>29.76</td>
</tr>
<tr>
<td>Program C</td>
<td>5</td>
<td>74.98</td>
<td>75.84</td>
</tr>
</tbody>
</table>

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, 1975]). 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.


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:
(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 Compatibility Requirements

Compatibility 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.)

2.0 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 Compatibility Requirements

Compatibility 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,nmmm
ASSIGN LINK TO <work-file-2>,NEW,SCRATCH,nmmm
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 rea-
sonable 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:


(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 siren 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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE-LINE</td>
<td>The current source line of COBOL text being scanned.</td>
</tr>
<tr>
<td>CONDITION-NAME-ENTRY</td>
<td>The structure containing a condition-name, its qualification hierarchy, and other associated information, which is retained in the SIRE-FILE.</td>
</tr>
<tr>
<td>SOURCE-WORD</td>
<td>The current source word of the COBOL text being scanned.</td>
</tr>
<tr>
<td>SAVE-WORD</td>
<td>Used to hold a copy of SOURCE-WORD.</td>
</tr>
<tr>
<td>QUALIFICATION-ENTRIES</td>
<td>Holds the qualification hierarchy of an item being scanned in the Procedure Division, for comparison with SIRE-FILE entries.</td>
</tr>
<tr>
<td>THE-TIME</td>
<td>The time of day, in hours, minutes, and seconds. Retrieved from the system.</td>
</tr>
<tr>
<td>HEADER</td>
<td>Information printed at top of each page in the listing output.</td>
</tr>
<tr>
<td>PRINT-LINE-1 through PRINT-LINE-8</td>
<td>These contain the formats of information to be placed in the listing output.</td>
</tr>
<tr>
<td>THE-DATE</td>
<td>The date, as retrieved from the system.</td>
</tr>
<tr>
<td>ACCUMULATIVE-COMPLEXITY</td>
<td>The total complexity of the COBOL source input program.</td>
</tr>
<tr>
<td>GO-TO-COMPLEXITY</td>
<td>Used as a temporary structure to retain complexity of potential GO-TO-DEPENDING situation.</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th><strong>LAST-LEVEL</strong></th>
<th>Level-number of last data-item scanned.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RUNNING-SECTION-COUNT</strong></td>
<td>The running total of the number of paragraphs in the current section being scanned.</td>
</tr>
<tr>
<td><strong>RUNNING-PROGRAM-COUNT</strong></td>
<td>The running total of the number of paragraphs in the source program.</td>
</tr>
<tr>
<td><strong>RUNNING-SECTION-TOTAL</strong></td>
<td>The running total of the complexity of the paragraphs of the section currently being scanned.</td>
</tr>
<tr>
<td><strong>RUNNING-PROGRAM-TOTAL</strong></td>
<td>The running total of the complexity of the paragraphs in the source program.</td>
</tr>
<tr>
<td><strong>SECTION-AVERAGE</strong></td>
<td>The average complexity of the paragraphs in the last section scanned.</td>
</tr>
<tr>
<td><strong>PROGRAM-AVERAGE</strong></td>
<td>The average complexity of the paragraphs in the source program.</td>
</tr>
<tr>
<td><strong>SIRE-NAME</strong></td>
<td>The id-name of a qualifier.</td>
</tr>
<tr>
<td><strong>FILE-FLAG</strong></td>
<td>A flag used to indicate whether the source program being scanned has fixed or variable length records.</td>
</tr>
<tr>
<td><strong>EOF-FLAG</strong></td>
<td>A flag used to indicate that the scanning of the source program has reached the end of the source file.</td>
</tr>
<tr>
<td><strong>AREA-A-FLAG</strong></td>
<td>A flag used to indicate whether the AREA-A of the current source line is blank or non-blank.</td>
</tr>
<tr>
<td><strong>CONDITION-FIND-FLAG</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>DONE-SW</strong></td>
<td>A general purpose flag.</td>
</tr>
<tr>
<td><strong>CONSUME-SW</strong></td>
<td>A flag used in the scanning of literals.</td>
</tr>
<tr>
<td><strong>QUOTE-SW</strong></td>
<td>A flag used in the scanning of literals.</td>
</tr>
<tr>
<td>CONDITIONAL-SITUATION-SW</td>
<td>A flag used to indicate that a conditional situation has been found in the current source line being scanned.</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NO-NOT-SW</td>
<td>A flag used to indicate the presence of 'NOT' in the current source line.</td>
</tr>
<tr>
<td>MORE-CONDITION-SW</td>
<td>A flag used to indicate whether the current conditional expression being scanned is compound or not.</td>
</tr>
<tr>
<td>RELATIONAL-OPERATOR-SW</td>
<td>A flag used to indicate the presence of a relational operator in the current source line being scanned.</td>
</tr>
<tr>
<td>GO-TO-DEPENDING-SW</td>
<td>A flag used to indicate that a GO-TO-DEPENDING has been found in the current source line being scanned.</td>
</tr>
<tr>
<td>FIRST-TIME-THRU-SW</td>
<td>A flag used to indicate whether or not this is the first module being scanned.</td>
</tr>
<tr>
<td>LINE-NUMBER</td>
<td>The line-number of the current source line being scanned.</td>
</tr>
<tr>
<td>LINE-NUMBER-DISPLAY</td>
<td>Used for conversion of numeric LINE-NUMBER to alphanumeric value for display to the CRT device.</td>
</tr>
<tr>
<td>ADD-ONE-SW</td>
<td>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.</td>
</tr>
<tr>
<td>PRINT-TOTAL-N-AVERAGE-SW</td>
<td>A flag used to indicate whether the situation exists where the total and average complexity should be printed.</td>
</tr>
</tbody>
</table>
4.2 Routines

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

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>Outermost routine (Driver).</td>
</tr>
<tr>
<td>GET-WORD</td>
<td>Isolate the next token in the COBOL source program.</td>
</tr>
<tr>
<td>READ-NEXT</td>
<td>Inputs the next character of the COBOL source file.</td>
</tr>
<tr>
<td>ISOLATE-WORD</td>
<td>Isolate a token in the COBOL source program.</td>
</tr>
<tr>
<td>CONSUME-LITERAL</td>
<td>Isolate a literal token in the COBOL text.</td>
</tr>
<tr>
<td>EXAMINE-AND-ISOLATE-SUBSCRIPTS</td>
<td>Strip off and examine subscripts.</td>
</tr>
<tr>
<td>GET-TO-TOP-OF-PAGE</td>
<td>Slew the output listing to the top of the page and print the heading with page number.</td>
</tr>
<tr>
<td>GET-TIME-AND-DATE</td>
<td>Accept time and date for listing from the system.</td>
</tr>
<tr>
<td>PROCESS-PROGRAM-ID-PARAGRAPH</td>
<td>Scan for program-id to put in the listing.</td>
</tr>
<tr>
<td>COLLECT-CONDITION-NAMES</td>
<td>Collect condition-name entries in the Environment and Data Divisions.</td>
</tr>
<tr>
<td>COLLECT-ANY-SWITCHES</td>
<td>Scan the COBOL source program, looking for any switches.</td>
</tr>
<tr>
<td>COLLECT-ANY-88-LEVELS</td>
<td>Scan the COBOL source program, collecting all the 88 level items found in the Data Division.</td>
</tr>
<tr>
<td>ANALYZE-CURRENT-LEVEL</td>
<td>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</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REGISTER-88-LEVEL</td>
<td>Scan an 88 level item and register it and its corresponding complexity.</td>
</tr>
<tr>
<td>OUTPUT-CONDITION-ENTRY</td>
<td>Output an entry to the SIRE-FIAT.</td>
</tr>
<tr>
<td>OUTPUT-SIRE-NAME</td>
<td>Output an entry to NAME-FIAT.</td>
</tr>
<tr>
<td>PROCESS-PROCEDURE-DIVISION</td>
<td>Analyze the complexity of the paragraphs found in the Procedure Division.</td>
</tr>
<tr>
<td>GET-MODULE-COMPLEXITY</td>
<td>Analyze the complexity of an individual paragraph.</td>
</tr>
<tr>
<td>FIND-CONDITIONAL-SITUATIONS</td>
<td>Scan the COBOL source program, looking for a conditional expression.</td>
</tr>
<tr>
<td>SCAN-FOR-GOTO-DEPENDING</td>
<td>For every possible GOTO DEPENDING situation, accumulate the potential</td>
</tr>
<tr>
<td></td>
<td>complexity. If the DEPENDING phrase is found, add to complexity totals</td>
</tr>
<tr>
<td></td>
<td>the accumulated complexity for the GOTO DEPENDING.</td>
</tr>
<tr>
<td>CHECK-MODULE-START</td>
<td>Scan the COBOL source program, looking for the start of a section or</td>
</tr>
<tr>
<td></td>
<td>paragraph.</td>
</tr>
<tr>
<td>ANALYZE-COMPLEXITY</td>
<td>Scan a conditional, calculating its complexity.</td>
</tr>
<tr>
<td>GET-PAST-AND-REMOVE-PARENS</td>
<td>Scan past parentheses.</td>
</tr>
<tr>
<td>SCAN-FOR-NOT</td>
<td>Scan for NOT, setting NO-NOT-SW as to whether one is found or not found.</td>
</tr>
<tr>
<td>SCAN-QUALIFICATION-N-SUBSCRIPT</td>
<td>Scan any qualification and associated subscripts, scanning past the</td>
</tr>
<tr>
<td></td>
<td>subscripts, and placing the qualification items in the table called</td>
</tr>
<tr>
<td></td>
<td>QUALIFICATION-ENTRIES.</td>
</tr>
<tr>
<td>SCAN-FOR-IS-N-NOT</td>
<td>Scan past IS and NOT if they are found.</td>
</tr>
<tr>
<td>SCAN-FOR-OR-N-AND</td>
<td>Scan for the connectors 'OR' and 'AND', which indicate compound</td>
</tr>
<tr>
<td></td>
<td>conditionals.</td>
</tr>
</tbody>
</table>
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-VALUES
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 600-1pm 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.)

2.0 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-alphabetics. 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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE-LINE</td>
<td>The current source line of COBOL text being scanned.</td>
</tr>
<tr>
<td>SOURCE-WORD</td>
<td>The current source word of the COBOL text being scanned.</td>
</tr>
<tr>
<td>SAVE-WORD</td>
<td>Used to hold a copy of SOURCE-WORD.</td>
</tr>
<tr>
<td>OPERATOR-ENTRIES</td>
<td>Table which holds the id-name of known COBOL operators and their frequencies.</td>
</tr>
<tr>
<td>ALPHABET-ENTRIES</td>
<td>Holds the links chaining together operand id-names using the first character of the operand id-names as an index.</td>
</tr>
<tr>
<td>OPERAND-HEAD-LINKS</td>
<td>The head links of the chain of link elements for operands and their frequencies.</td>
</tr>
<tr>
<td>OPERAND-ENTRY</td>
<td>Contains an operand id-name and its frequency.</td>
</tr>
<tr>
<td>THE-TIME</td>
<td>The time of day, in hours, minutes, and seconds. Retrieved from the system.</td>
</tr>
<tr>
<td>HEADER</td>
<td>Information printed at top of each page in listing output.</td>
</tr>
<tr>
<td>PRINT-LINE-1 through PRINT-LINE-20</td>
<td>These contain the formats of information to be placed in the listing output.</td>
</tr>
<tr>
<td>THE-DATE</td>
<td>The date, as retrieved from the system.</td>
</tr>
<tr>
<td>PAGE-COUNT</td>
<td>Counter incremented for each page of the listing output.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PTR</td>
<td>Index into the source line being scanned.</td>
</tr>
<tr>
<td>START-OF-WORD</td>
<td>Index indicating where the starting position in source line of the current word being scanned is.</td>
</tr>
<tr>
<td>END-OF-WORD</td>
<td>Index indicating position of end character in the source line of the current work being scanned.</td>
</tr>
<tr>
<td>WORD-SIZE</td>
<td>Size in characters of the current word being scanned.</td>
</tr>
<tr>
<td>SOURCE-PTR</td>
<td>Index into the source line used in scanning literals.</td>
</tr>
<tr>
<td>PTR-TEMP</td>
<td>Temporary variable to hold indices into the current source line being scanned.</td>
</tr>
<tr>
<td>SUBSCRIPT-TEMP</td>
<td>Index used for indicating position of the end of the source word being scanned when skipping past parentheses in the Procedure Division.</td>
</tr>
<tr>
<td>DELIMITER-TEMP</td>
<td>Temporary value used to hold source word delimiters in the scanning of subscripts.</td>
</tr>
<tr>
<td>SOURCE-WORD-TEMP</td>
<td>Temporary storage for retaining source word values.</td>
</tr>
<tr>
<td>SOURCE-STOP</td>
<td>Contains the delimiter of the current source word being scanned.</td>
</tr>
<tr>
<td>SAVE-STOP</td>
<td>Used to hold a copy of SOURCE-STOP.</td>
</tr>
<tr>
<td>LINE-NUMBER</td>
<td>The line-number of the current source line being scanned.</td>
</tr>
<tr>
<td>LINE-NUMBER-DISPLAY</td>
<td>Used for conversion of numeric LINE-NUMBER to alphanumeric value for display to the CRT device.</td>
</tr>
<tr>
<td>OPERAND-KEY</td>
<td>Relative key for the OPERAND scratch work file.</td>
</tr>
<tr>
<td>LINK-KEY</td>
<td>Relative key for the LINK scratch work file.</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLIER</td>
<td>Used in the calculation of base two logarithms.</td>
</tr>
<tr>
<td>FRACTION</td>
<td>Used in the calculation of base two logarithms.</td>
</tr>
<tr>
<td>I</td>
<td>Index used in the calculation of base two logarithms.</td>
</tr>
<tr>
<td>X-PLUS-1</td>
<td>Used in the calculation of base two logarithms.</td>
</tr>
<tr>
<td>X-MINUS-1</td>
<td>Used in the calculation of base two logarithms.</td>
</tr>
<tr>
<td>CONVERSION-VALUE</td>
<td>The value by which a natural logarithm is multiplied to arrive at an equivalent base two logarithm.</td>
</tr>
<tr>
<td>FILE-FLAG</td>
<td>A flag used to indicate whether the source program being scanned has fixed or variable length records.</td>
</tr>
<tr>
<td>EOF-FLAG</td>
<td>A flag used to indicate that the scanning of the source program has reached the end of the source file.</td>
</tr>
<tr>
<td>AREA-A-FLAG</td>
<td>A flag used to indicate whether the AREA-A of the current source line is blank or non-blank.</td>
</tr>
<tr>
<td>CONSUME-SW</td>
<td>A flag used in the scanning of literals.</td>
</tr>
<tr>
<td>QUOTE-SW</td>
<td>A flag used in the scanning of literals.</td>
</tr>
<tr>
<td>DONE-SW</td>
<td>A general purpose flag.</td>
</tr>
<tr>
<td>OPERAND-FIND-FLAG</td>
<td>A flag used to indicate whether the current operand being scanned has been previously encountered and thus been placed in the appropriate link list.</td>
</tr>
</tbody>
</table>
### 4.2 Routines

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

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

5.1 Development Systems

The primary development system used for Halstead's Metrics Evaluation Tool was an I2270 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.
170 /  
200 INPUT-OUTPUT SECTION.  
210 FILE-CONTROL.  
220 SELECT FIXED-INPUT ASSIGN TO DISC.  
230 SELECT VARIABLE-INPUT ASSIGN TO DISC.  
240 SELECT PRINT-FILE ASSIGN TO PRINTER.  
250 SELECT SIRE-FILE ASSIGN TO DISC.  
260 ORGANIZATION IS RELATIVE.  
270 ACCESS MODE IS RANDOM.  
280 RELATIVE KEY IS SIRE-KEY.  
290 SELECT NAME-FILE ASSIGN TO DISC  
300 ORGANIZATION IS RELATIVE.  
310 ACCESS MODE IS RANDOM.  
320 RELATIVE KEY IS NAME-KEY.  
330  
340 I-O-CONTROL.  
350 SAME RECORD AREA FOR FIXED-INPUT, VARIABLE-INPUT.  
360 DATA DIVISION.  
370 FILE SECTION.  
380 FD FIXED-INPUT  
390 BLOCK CONTAINS 60 RECORDS  
400 RECORD CONTAINS 80 CHARACTERS  
410 LABEL RECORDS ARE STANDARD  
420 VALUE OF FILE-ID IS "I".  
430 01 INPUT-LINE PIC X(80).  
440  
450 FD VARIABLE-INPUT  
460 BLOCK CONTAINS 512 CHARACTERS  
470 RECORD VARYING 1 TO 80 CHARACTERS DEPENDING ON ID-1  
480 LABEL RECORDS ARE STANDARD  
490 VALUE OF FILE-ID IS "I".  
500 01 INPUT-LINE PIC X(80).  
510  
520 FD PRINT-FILE  
530 BLOCK CONTAINS 1 RECORDS  
540 RECORD CONTAINS 132 CHARACTERS  
550 LABEL RECORDS ARE OMITTED  
560 LINADL IS 56 LINES.  
570 LINES AT TOP 3.  
580 LINES AT BOTTOM 5.  
590 01 PRINT-RECORD PIC X(132).  
600  
610 FD SIRE-FILE  
620 LABEL RECORDS ARE STANDARD.  
630 01 SIRE-RECORD.  
640 05 S-COUNT-NAME PIC X(30).  
650 05 S-TABLE PIC X(30).  
660 05 S-TABLE-SIZE PIC 99.  
670 05 S-SINGLE-VALUE PIC 9(18).  
680 05 S-SINGLE-NAME-KEY PIC X(30).  
690 05 S-SINGLE-LEVEL PIC X(5).  
700  
710 FD NAME-FILE  
720 BLOCK CONTAINS 17 RECORDS  
730 LABEL RECORDS ARE STANDARD.  
740 01 NAME-RECORD PIC X(30).
WORKING-STORAGE SECTION.

01 SOURCE-LINE.
  05 SQUENCE-NUMBER PIC X(6).
  05 COMMENT-FIELD PIC X.
  04 COMMENT-ON-DEBUG VALUES ARE "*", "D", "/".
  08 LINE-CONTINUATION VALUE IS "-".
  03 TEXT-LINE PIC X(8).
  04 ID-FIELD PIC X(8).
  05 SOURCE-LINE-REDEFINED REDEFINES SOURCE-LINE.
  05 SOURCE-ITEMS OCCURS 80 TIMES.
  10 SOURCE-LINE-ITEM PIC X.

01 CONDITION-NAME-ENTRY.
  05 CONDITION-NAME PIC X(30).
  05 SIZE-FAULT OCCURS 50 TIMES.
  10 SIZE-NAME-POINTER PIC 9(4).
  10 SIZE-LEVEL PIC X(2).
  05 COMPLEXITY-VALUE PIC 9(3).
  05 SWITCH-FLAG PIC 9.
  08 IS-A-SWITCH VALUE 1.
  08 NOT-A-SWITCH VALUE 0.
  05 TABLE-SIZE PIC 99.

01 SOURCE-WORD PIC X(200).
  08 A-LEVEL-NUMBER VALUES ARE "77", "88", "FD".
  08 "Qi" THRU "66".
  08 RELATIONAL-OPERATOR VALUES ARE <", ">", "=".
  08 GREATER", "LESS", "EQUAL".
  08 A-SWITCH VALUES ARE "SWITCH-1" THRU "SWITCH-99".
  08 CLASS-CONDITION VALUES ARE "POSITIVE", "NEGATIVE", "ZERO", "NUMERIC", "ALPHANUMERIC", "ALPHABETIC".
  01 SOURCE-WORD-REDEFINED REDEFINES SOURCE-WORD.
  05 SOURCE-WORDS OCCURS 120 TIMES.
  10 SOURCE-WORD-ITEM PIC X.

01 SAVE-WORD PIC X(30).

01 QUALIFICATION-ENTRIES.
  05 QUALIFICATION-TABLE OCCURS 50 TIMES.
  10 QUALIFICATION-ITEM PIC X(30).

01 THE-TIME.
  05 MINUTES PIC 99.
  05 SECONDS PIC 99.
01 HEADER.
  05 FILLER PIC X(8) VALUE IS 'MCADE'S'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS 'COMPLEXITY'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS 'MEASURE'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS 'PROGRAM-NAME'.
  05 FILLER PIC X VALUE IS '""'.
  05 PROGRAM-NAME PIC X(30).
  05 FILLER PIC X(6) VALUE IS SPACES.
  05 DATE-OUT PIC 999999.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS SPACES.
  05 MINUTES-OUT PIC 99.
  05 FILLER PIC X VALUE IS '""'.
  05 SECONDS-OUT PIC 99.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.
  05 PAGE-NUMBER PIC X(13).

01 PRINT-LINE-1.
  05 FILLER PIC X VALUE SPACES.
  05 FILLER PIC X VALUE IS 'SECTION-NAME'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS 'SECTION-NAME'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.

01 PRINT-LINE-2.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS 'SECTION-NAME'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS 'SECTION-NAME'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS '""'.

01 PRINT-LINE-3.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS 'PARAGRAPH-NAME'.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS '""'.
  05 FILLER PIC X VALUE IS SPACES.
01 PRINT-LINE-B.
  05 FILLER PIC X(10) VALUE IS SPACES.
  05 FILLER PIC X(7) VALUE IS "AVERAGE".
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X(7) VALUE IS "COMPLEXITY".
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X(3) VALUE IS "FOR".
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X(4) VALUE IS "THIS".
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X(7) VALUE IS "PROGRAM".
  05 FILLER PIC X VALUE IS SPACES.
  05 FILLER PIC X VALUE IS "=".
  05 FILLER PIC X VALUE IS SPACES.
  05 PROGRAM-AVERAGE-OUT PIC Z(14).9(4).
2440 77 THE-DATE PIC 9(4).
2441 77 ACUMULATIVE-COMPLEXITY PIC 9(14).
2442 77 GO-YU-COMPLEXITY PIC 9(14).
2443 77 QUALIFICATION-COUNT PIC 9(1).
2450 77 PAGE-COUNT PIC 9(1).
2460 77 NUMBER-OF-CONDITION-NAMES PIC 9(4).
2470 77 SENSE-KEY PIC 9(16).
2480 77 NAME-KEY PIC 9(16).
2490 77 PKH PIC 9(16).
2500 77 START-OF-WORD PIC 9(16).
2510 77 END-OF-WORD PIC 9(16).
2520 77 WORD-SIZE PIC 9(16).
2530 77 SOURCE-PTR PIC 9(16).
2540 77 PTY-TEMP PIC 9(16).
2550 77 DELLIMITER-TEMP PIC 9(16).
2560 77 SOURCE-START PIC 9(16).
2570 77 ID-1 PIC 9(9).
2580 77 SIZE-TABLE-SUBSCRIPT PIC 9.
2590 77 SUBSCRIPT-TEMP PIC 9(1).
2600 77 CURRENT-LEVEL PIC 9.
2610 77 LAST-LEVEL PIC 9.
2620 77 RUNNING-SECTION-COUNT PIC 9(19).
2630 77 RUNNING-PROGRAM-COUNT PIC 9(19).
2640 77 RUNNING-SECTION-TOTAL PIC 9(19).
2650 77 RUNNING-PROGRAM-TOTAL PIC 9(19).
2660 77 SECT-SECTION-AVERAGE PIC 9(4).VALUE IS ZERO.
2670 77 PROGRAM-AVERAGE PIC 9(4).VALUE IS ZERO.
LINE  DEBJG  SEQ/NO  A...B...........................10......

312*       / PROCEDURE DIVISION.
314*       *
315*       DECLARATIVES.*
316*       *
317*000002  OPEN-ERROR SECTION.
318*       USE AFTER STANDARD EXCEPTION PROCEDURE ON FIXED-INPUT.
319*
320*       *
321*000002  OPEN-VARIABLE.
321*000002  OPEN INPUT VARIABLE-INPUT.
322*000006  MOVE 1 TO FILE-FLAG.
323*000006  OPEN-ENO.*
324*       *
325*       END DECLARATIVES.
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

1 MAIN ROUTINE

2 GET TIME AND DATE ROUTINE

3 PROCESS PROGRAM ID PARAGRAPH ROUTINE

4 COLLECT CONDITION NAME ROUTINE
   4.1 COLLECT ANY SWITCHES
   4.2 COLLECT ANY RA LEVELS
   4.2.1 ANALYZE CURRENT LEVEL
   4.2.2 REGISTER RA LEVEL
   4.2.2.1 OUTPUT CONDITION ENTRY
   4.2.2.2 OUTPUT SIRE NAME

5 PROCESS PROCEDURE DIVISION ROUTINE
   5.1 GET MODULE COMPLEXITY
   5.1.1 FIND CONDITIONAL SITUATIONS
   5.1.1.1 SCAN FOR GO TO DEPENDINGS
   5.1.2 CHECK MODUL START
   5.1.3 ANALYZE COMPLEXITY
   5.1.3.1 GET PAST AND REMOVE PARENS
   5.1.3.2 SCAN FOR NOT
   5.1.3.3 SCAN QUALIFICATION AND SUBSCRIPT
   5.1.3.4 SCAN FOR IS AND NOT
   5.1.3.5 SCAN FOR OR AND AND
   5.1.3.6 CHECK IF RELATIONAL OPERATOR
   5.1.3.7 CHECK OUT CONDITION
   5.1.3.7.1 GO THRU CONDITION NAMES

6 PRINT ROUTINES
1 MAIN ROUTINE

FUNCTION: OUTRODUCTION (DRIVER)

INPUT: NUN

OUTPUT: NUN

LOCAL DATA: FIXED-LENGTH-FILE

CALLED BY: NUN

CALLS: GET-TIME-AND-DATE

PRINT-SECTION TOTAL-N-AVERAGE

PRINT-PROGRAM TOTAL-N-AVERAGE

-----------------------------------------------------------------------------------

MAIN-PROGRAM SECTION

MAIN:

PERFORM GET-TIME-AND-DATE.

OPEN INPUT FIXED-INPUT.

OPEN OUTPUT PRINT-FILE.

PERFORM PROCESS-PROGRAM-ID-PARAGRAPH.

PERFORM COLLECT-CONDITION-NAMES.

PERFORM PROCESS-PROCEDURE-DIVISION.

PERFORM PRINT-SECTION TOTAL-N-AVERAGE.

PERFORM PRINT-PROGRAM TOTAL-N-AVERAGE.

IF FIXED-LENGTH-FILE

CLOSE FIXED-INPUT.

CLOSE PRINT-FILE.

STOP RUN.
FUNCTION: READ NEXT ROUTINE

INPUT: NONE

OUTPUT: PTR

LOCAL DATA: LINE-NUMBER

CALLED BY: GET-WORD-actual

CALLS: NONE

READ-NEXT:

MOVE SPACES TO INPUT-LINE OF VARIABLE-INPUT.

IF FIXED-LENGTH-FIELD
READ FIXED-INPUT RECORD
ELSE READ VARIABLE-INPUT RECORD

AT END MOVE 1 TO EOF-FLAG

MOVE 0 TO PTR.

MOVE 1 TO INPUT-LINE OF VARIABLE-INPUT TO SOURCE-LINE.

DISPLAY SOURCE-LINE.

ADD 1 TO LINE-NUMBER.

MOVE LINE-NUMBER TO LINE-NUMBER-DISPLAY.

DISPLAY "SOURCE LINE NUMBER = ", LINE-NUMBER-DISPLAY.

LINE 20 POSITION 22.
FUNCTION: ISOLATE A TOKEN IN THE COBOL SOURCE

INPUT: PTR
OUTPUT: SOURCE-LINE

LOCAL DATA: START-OF-WORD
CALLED BY: GET-WORD-ACTUAL
CALLS: CONSUME-LITERAL
EXAMINE-AND-ISOLATE-SUBSCRIPTS

MOVE SPACES TO SOURCE-WORD.
MOVE PTR TO START-OF-WORD.
IF SOURCE-LINE-ITEM (PTR) = "":
PERFORM CONSUME-LITERAL.
ELSE UNSTRING SOURCE-LINE.
DELIMITED BY "," OR ";" OR ":" OR ALL SPACES.
INTO SOURCE-WORD.
DELIMITED IN SOURCE-STOP.
COUNT IN WORD-SIZE.
WITH POINTER PTR.
PERFORM EXAMINE-AND-ISOLATE-SUBSCRIPTS.

IF START-OF-WORD IS LESS THAN 12
AND SOURCE-WORD-ITEM1 NOT EQUAL "":
AND SOURCE-WORD NOT EQUAL SPACES.
MOVE 1 TO AREA-A-FLAG.
ELSE MOVE ZERO TO AREA-A-FLAG.

DISPLAY SPACES, LINE 7, SIZE 40.
DISPLAY "":
LINE 7, POSITION PTR.
SOURCE-WORD.
LINE 9, POSITION 22.
**FUNCTION**: ISOLATE A LITERAL TOKEN IN THE COBOL TEXT

**INPUT**: PTR

**OUTPUT**: SOURCE-WORD

**LOCAL DATA**: QUOTE-SW

**CALLED BY**: ISOLATE-WORD

**CALLS**: READ-NEXT

CONSUME-LITERAL

MOVE ZERO TO Source-PTR.

MOVE PTR TO CONSUME-SW.

MOVE 1 TO QUOTE-SW.

PERFORM CONSUME-LITERAL-CHARACTER UNTIL CONSUME-DONE.

PERFORM ABSORB-SPACES UNTIL Source-LINE-ITEM (PTR) NOT = SPACES.

CONSUME-LITERAL-CHARACTER.

ADD 1 TO Source-PTR.

IF PTR GREATER THAN #2

PERFORM READ-NEXT.

PERFORM FLIP-QUOTE-SW.

UNNEST NOT COMMENT-OR-DEBUG.

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 #2

PERFORM READ-NEXT.

PERFORM READ-NEXT.

PERFORM READ-NEXT.

UNTIL NOT COMMENT-OR-DEBUG.

ADD 1 TO PTR.

FLIP-QUOTE-SW.
**Q.1.2.2 EXAMINE AND ISOLATE SUBSCRIPTS ROUTINE**

**FUNCTION:** STRIP OFF AND EXAMINE SUBSCRIPTS

**INPUT:** WORD-SIZE

**OUTPUT:** SOURCE-WORD

**LOCAL DATA:** END-OF-WORD

**CALLED BY:** ISOLATE-WORD

**CALLS:** NONE

---

**640**

```assembly
FUNCTION: STRIP OFF AND EXAMINE SUBSCRIPTS

INPUT: WORD-SIZE

OUTPUT: SOURCE-WORD

LOCAL DATA: END-OF-WORD

CALLED BY: ISOLATE-WORD

CALLS: NONE
```

---

**641**

```assembly
641+0001F0 EXAMINE-AND-ISOLATE-SUBSCRIPTS
642+0001F0 ADD WORD-SIZE START-OF-WORD
642+0001F0 GIVING END-OF-WORD
642+0001F0 SUBTRACT 1 FROM END-OF-WORD
642+0001F0 GIVING END-OF-WORD
```

---

**640**

```assembly
640+000000 IF SOURCE-LINE-ITEM (END-OF-WORD) = ""
640+000010 AND SOURCE-ITEM (I) NOT EQUAL "("
640+000020 AND SOURCE-ITEM (I) NOT EQUAL ")"
640+000030 UNSTRING SOURCE-WORD
640+000040 DELIMITED BY "(" OR SPACES
640+000050 INTO SOURCE-WORD-TEMP
640+000060 DELIMITER IN DELIMITER-TEMP
640+000070 WITH POINTER PTR-TEMP
640+000080 IF DELIMITER-TEMP = ""
640+000090 PERFORM BACK-UP-TO-BEFORE-SUBSCRIPT
640+000100 UNTIL SOURCE-LINE-ITEM (END-OF-WORD) = ""
640+000110 PERFORM BACK-UP-TO-BEFORE-SUBSCRIPT
```

---

**640**

```assembly
640+000000 MOVE END-OF-WORD TO PTR
640+000000 PERFORM BACK-UP-TO-BEFORE-SUBSCRIPT
```

---

**640**

```assembly
640+000000 BACK-UP-TO-BEFORE-SUBSCRIPT
640+000000 MOVE SPACES TO SOURCE-WORD-ITEM (WORD-SIZE)
640+000000 SUBTRACT 1 FROM WORD-SIZE
640+000000 SUBTRACT 1 FROM END-OF-WORD
```

---
//
// ************
// 01  GET TO TOP OF PAGE
// 01  FUNCTION: SLEW THE OUTPUT LISTING TO THE TOP OF THE PAGE AND PRINT THE HEADING WITH PAGE NUMBER
// 01  INPUT: PAGE-COUNT
// 01  HEADER
// 01  OUTPUT: PAGE-COUNT
// 01  LOCAL DATA: PRINT-RECORD
// 01  CALLED BY: PRINT-SECTION-NAMESPACE-N-NUMBER
// 01  PRINT-SECTION-NAME
// 01  PRINT-MODULE-NAME
// 01  PRINT-COMPLEXITY
// 01  PRINT-SECTION-TOTAL-N-AVERAGE
// 01  PRINT-PROGRAM-TOTAL-N-AVERAGE
// 01  CALLS: NONE
// 01
// 663  GET TO TOP-OF-PAGE.
// 669  ADD 1 TO PAGE-COUNT.
// 669  MOVE PAGE-COUNT TO PAGE-NUMBER.
// 667  WRITE PRINT-RECORD FROM HEADER AFTER ADVANCING PAGE.
669* 1
669* 2
670* 3
671* 4
672* 5
673* 6
674* 7
675* 8
676* 9
677* 10
678* 11
679* 12
680* 13
681* 14
682* 15
683* 16
684* 17
685*01028D 18
686* 19
687*01028D 20
688*01028D 21
689*01028D 22
690*01028C 23
691*01028C 24
692*01028C 25
693*01028C 26

669*Z

673* FUNCTION ACCEPT TIME AND DATE FOR LISTING

675* INPUT TIME AND DATE FROM SYSTEM

676* OUTPUT DATE-OUT

677* HOURS-OUT

678* MINUTES-OUT

679* SECONDS-OUT

680* LOCAL DATA: THE-TIME

681* THE-DATE

682* CALLED BY: MAIN

683* CALLS: NONE

685*01028D

688*01028D

689*01028D

690*01028C

691*01028C

692*01028C

693*01028C

TIME-AND-DATE SECTION 90.

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.
3 PROCESS PROGRAM-ID PARAGRAPH ROUTINE

FUNCTION: SCAN FOR PROGRAM-ID TO PUT IN THE LISTING

INPUT: SOURCE-WORD

OUTPUT: PAGE-COUNT

LOCAL DATA: NONE

CALLED BY: MAIN

CALLS: GET-WORD

FUNCTIONS:

7000 7000 7000
7010 7010 7010
7020 7020 7020
7030 7030 7030
7040 7040 7040
7050 7050 7050
7060 7060 7060
7070 7070 7070
7080 7080 7080
7090 7090 7090
7100 7100 7100
7110 7110 7110
7120 7120 7120
7130 7130 7130
7140 7140 7140
7150 7150 7150
7160 7160 7160
7170 7170 7170
7180 7180 7180
7190 7190 7190
7200 7200 7200
7210 7210 7210
7220 7220 7220
7230 7230 7230
7240 7240 7240
7250 7250 7250
7260 7260 7260
7270 7270 7270
7280 7280 7280
7290 7290 7290
7300 7300 7300
COLLECT CONDITION NAME ROUTINE

FUNCTION
THE ENVIRONMENT AND DATA DIVISIONS

INPUT SOURCE-WORD
OUTPUT SINK-FILE
LOCAL DATA NAME-FILE
CALLED BY MAIN
CALLS GET-WORD
CALLS COLLECT-ANY-SWITCHES
CALLS COLLECT-ANY-BR-LEVELS

CONDITION-NAMING SECTION 92.

OPEN OUTPUT SINK-FILE.
OPEN NAME-FILE.
PERFORM GET-WORD UNTIL SOURCE-WORD = "SPECIAL-NAMES" OR SOURCE-WORD = "DATA" OR SOURCE-WORD = "PROCEDURE".
IF SOURCE-WORD = "SPECIAL-NAMES" PERFORM COLLECT-ANY-SWITCHES UNTIL SOURCE-WORD = "DATA" OR SOURCE-WORD = "PROCEDURE".
PERFORM COLLECT-ANY-BR-LEVELS UNTIL SOURCE-WORD = "PROCEDURE".
CLOSE SINK-FILE, NAME-FILE.
741.  /************
742.  ************
743.  4.1 COLLECT ANY SWITCHES
744.  ************
745.  ************
746.  FUNCTION: SCAN THE COBOL SOURCE PROGRAM LOOKING FOR
747.  ************
748.  ************
749.  Any Switches.
750.  ************
751.  ************
752.  Input: SOURCE-WORD
753.  ************
754.  ************
755.  Output: SOURCE-WORD
756.  ************
757.  ************
758.  Local Data: None
759.  ************
760.  ************
761.  Called By: COLLECT-CONDITION-NAMES
762.  ************
763.  ************
764.  Calls: OUTPUT-CONDITION-ENTRY
765.  ************
766.  ************
767.  
768.  777* D302FE
769.  COLLECT-ANY-SWITCHES.
770.  778* D302FE
771.  PERFORM GET-WORD
772.  UNTIL A-SWITCH
773.  3X SOURCE-WORD = "DATA"
774.  OR SOURCE-WORD = "PROCEDURE"
775.  776.  777.  IF A-SWITCH
778.  PERFORM REGISTER-SWITCH
779.  PERFORM GET-WORD
780.  781.  IF SOURCE-WORD = "ON" OR SOURCE-WORD = "OFF"
782.  PERFORM REGISTER-SWITCH.
783.  784.  REGISTER-SWITCH.
785.  786.  MOVE ZERIES TO CONDITION-NAME-ENTRY.
787.  788.  789.  790.  PERFORM GET-WORD
791.  UNTIL SOURCE-WORD = "15".
792.  793.  794.  MOVE SOURCE-WORD TO CONDITION-NAME.
795.  796.  MOVE 1 TO COMPLEXITY-VALUE, SWITCH-FLAG.
797.  798.  PERFORM OUTPUT-CONDITION-ENTRY.
**COLLECT-ANY-88-LEVELS**

1. **PERFORM GET-WORD**
2. **UNTIL AREA-A-ocupied AND**
3. **END-WORD = "FD"**
4. **OR SOURCE-WORD = "01"**
5. **OR SOURCE-WORD = "78"**
6. **FOR SOURCE-WORD = "PROCEDURE"**
7. **MOVW "00" TO SOURCE-WORD**
8. **IF SOURCE-WORD NOT EQUAL "PROCEDURE"**
9. **MOVZ TO EA-TABLE-ENTRY**
10. **PERFORM REGISTER-CURRENT-LEVEL**
11. **PERFORM LOOK-FOR-88-LEVEL**
12. **UNTIL DONE**

**LOOK-FOR-88-LEVEL**

1. **MOVE CURRENT-LEVEL TO LAST-LEVEL**
2. **PERFORM GET-WORD**
3. **UNTIL SOURCE-STOP = ","**
4. **PERFORM GET-WORD**
5. **UNTIL A-LEVEL-NUMBER**
6. **OR SOURCE-WORD = "PROCEDURE"**
7. **MOVX "01" TO DONE-SW**
8. **ELSE IF SOURCE-WORD = "88"**
9. **PERFORM REGISTER-88-LEVEL**
10. **ELSE IF SOURCE-WORD NOT EQUAL "FD"**
11. **AND SOURCE-WORD NOT EQUAL "78"**
12. **AND SOURCE-WORD NOT EQUAL "01" AND**
13. **LAST-LEVEL NOT EQUAL "00"**
14. **PERFORM ANALYZE-CURRENT-LEVEL**
#520  THRU REGISTER=CURRENT-LEVEL
#530  ELSE MOVE "1" TO DONE-SW.

INPUT: SOURCE-WORD
       LAST-LEVEL
       SIRE-TABLE
       SIRE-TABLE-SUBSCRIPT
       NAME-KEY

OUTPUT: SIRE-TABLE
        SIRE-TABLE-SUBSCRIPT
        NAME-KEY
        CURRENT-LEVEL
        SIRE-NAME

LOCAL DATA: NONE
CALLED BY: LOOK-FOR-08-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-OK.
 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 00 TO -1.
 UNTIL SUBSCRIPT-TEMP EQUAL SIRE-TABLE-SUBSCRIPT.

DECREMENT-SUBSCRIPT.
 SUBTRACT 1 FROM SIRE-TABLE-SUBSCRIPT.
4.2.2 REGISTER 88 LEVEL

FUNCTION: SCAN AN 88 LEVEL ITCH AND REGISTER IT AND ITS CORRESPONDING COMPLEXITY

INPUT: SOURCE-WORD

OUTPUT: SICE-TABLE-SUBSCRIPT

TABLE-SIZE

COMPLEXITY-VALUE

LOCAL DATA: NONE

CALLED BY: COLLECT-ANY-88-LEVELS

CALLS: GET-WORD

OUTPUT-CONDITION-ENTRY

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

REGISTER 88-LEVEL.

EXECUTE GET-WORD.

MOVE SOURCE-WORD TO CONDITION-NAME.

MOVE SICE-TABLE-SUBSCRIPT TO TABLE-SIZE.

PERFORM CALCULATE-CONDITION-COMPLEXITY.

PERFORM OUTPUT-CONDITION-ENTRY.

CALCULATE-CONDITION-COMPLEXITY.

PERFORM GET-WORD.

PERFORM GET-WORD.

PERFORM GET-WORD.

PERFORM GET-WORD.

UNTIL SOURCE-WORD = "VALUE".

UNTIL SOURCE-WORD = "VALUES".

UNTIL SOURCE-WORD = "TRUE".

UNTIL SOURCE-WORD = "FALSE".

UNTIL SOURCE-WORD NOT EQUAL "TRUE".

AND SOURCE-WORD NOT EQUAL "FALSE".

MOVE 1 TO COMPLEXITY-VALUE.

PERFORM ACCUMULATE-COMPLEXITY.

PERFORM GET-WORD.

PERFORM GET-WORD.

IF SOURCE-WORD NOT EQUAL "THRU"

AND SOURCE-WORD NOT EQUAL "THROUGH"

ADD 1 TO COMPLEXITY-VALUE.
4.2.2.1 OUTPUT CONDITION ENTRY

FUNCTION OUTPUT AN ENTRY TO THE SIRE-FILE

INPUT CONDITION-NAME-ENTRY

OUTPUT SIRE-KEY

NUMBER-OF-CONDITION-NAMES

LOCAL DATA NONE

REGISTER-SWITCH

REGISTER-88-LEVEL

CALLS NONE

OUTPUT-CONDITION-ENTRY.
WRITE SIRE-RECORD FROM CONDITION-ENTRY.
INVALID KLY DISPLAY "XRN OUT OF SIRE-FILE SPACE"
STOP RUN.
ADD 1 TO NUMBER-OF-CONDITION-NAMES.
//
// 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

0204030533 OUTPUT-SIRE-NAME
0205030533 WRITE NAME-RECORD FROM SIRE-NAME
0206030533 INVALID KEY DISPLAY "RAN OUT OF NAME-FILE SPACE"
0207030533 STOP RUN.
1000 * PROCEDURE DIVISION ROUTINE
1001 * FUNCTION: ANALYZE THE COMPLEXITY OF THE PARAGRAPHS
1002 * FOUND IN THE PROCEDURE DIVISION
1003 * INPUT: NONE
1004 * OUTPUT: SINK-FILE
1005 * NAME-FILE
1006
1007 * LOCAL DATA: NONE
1008 * CALLED BY: MAIN
1009 * CALLS: GET-MODULE-COMPLEXITY
1010 * PRINT-COMPLEXITY
1011
1012 040280 PROCEDURE-DIVISION ROUTINES SECTION 53.
1013
1014 040280 PROCEDURE-DIVISION.
1015
1016 040280 OPEN INPUT SINK-FILE,
1017 040280 NAME-FILE.
1018 040280 PERFORM GET-MODULE-COMPLEXITY
1019 040280 UNTIL END-OF-FILE.
1020 040280 PERFORM PRINT-COMPLEXITY.
1021 040280 CLOSE SINK-FILE.
1022 040280 NAME-FILE.
1023
/* Get module complexity */

FUNCTION: ANALYZE THE COMPLEXITY OF AN INDIVIDUAL PARAGRAPH

INPUT: SIRE-FILE, NAME-FILE

OUTPUT: NONE

LOCAL DATA: CONDITIONAL-SITUATION-SW

CALLED BY: GET-MODULE-COMPLEXITY

CALLS: FIND-CONDITIONAL-SITUATIONS, CHECK-MODULE-START

GET-MODULE-COMPLEXITY.

MOVE ZERO TO CONDITIONAL-SITUATION-SW.

PERFORM FIND-CONDITIONAL-SITUATIONS UNTIL END-OF-FILE

OR AREA-A-CONDITIONAL-SITUATION

OR AREA-A-OCCUPIED.

IF NOT END-OF-FILE

IF AREA-A-OCCUPIED

PERFORM CHECK-MODULE-START

ELSE IF AREA-A-CONDITIONAL-SITUATION

PERFORM ANALYZE-COMPLEXITY.
FUNCTION: SCAN THE COBOL SOURCE PROGRAM, LOOKING FOR A CONDITIONAL.

INPUT: SOURCE-WORD

OUTPUT: SOURCE-WORD

ACCUMULATIVE-COMPLEXITY

RUNNING-SECTION-TOTAL

RUNNING-PROGRAM-TOTAL

LOCAL DATA: TIME-TO-ADD-UNE-TO-COMPLEXITY

GO-TO-DEPENDING-SW

GO-TO-COMPLEXITY

CALLED BY: GET-MODULE-COMPLEXITY

CALLED: GET-WORD

SCAN-FOR-GOTO-DEPENDING

***************
1077...040334  FIND-CONDITIONAL-SITUATIONS.
1079...040334  IF TIME-TO-ADD-UNE-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-UNE-SW.
1086...040338  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...040338  IF NOT-END-OF-FILE
1095... IF SOURCE-WORD = "AY"
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...040338  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...04037E  IF NOT-END-OF-FILE
1110... PERFORM GET-WORD.
1111... D DISPLAY ACCUMULATIVE-COMPLEXITY.
1112... D LINE 15, POSITION 34.
FUNCTION 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

INPUTS
SOURCE-WORD
ACUMULATIVE-COMPLEXITY
RUNNING-SECTION-TOTAL
RUNNING-PROGRAM-TOTAL
GOTO-COMPLEXITY
GOTO-DEPENDING-SW
ACUMULATIVE-COMPLEXITY
LOCAL DATA: NONE
CALLED BY: FIND-CONDITIONAL-SITUATIONS
CALLS: NONE

SCAN-FOR-GOTO-DEPENDING

IF SOURCE-WORD = "GO"
    MOVE 1 TO GO-TO-DEPENDING-SW
ELSE IF SOURCE-WORD = "DEPENDING"
    MOVE ZERO TO GO-TO-DEPENDING-SW
    ADD GO-TO-COMPLEXITY TO ACCUMULATIVE-COMPLEXITY
    ADD GO-TO-COMPLEXITY TO RUNNING-SECTION-TOTAL
    ADD GO-TO-COMPLEXITY TO RUNNING-PROGRAM-TOTAL
    CALL GO-TO-DEPENDING-SW
    AND SOURCE-WORD NOT EQUAL "TO"
    ADD 1 TO GO-TO-COMPLEXITY.
FUNCTION: SCAN THE COBOL SOURCE, LOOKING FOR THE START OF A SECTION OR PARAGRAPH.

INPUT:
- SOURCE-WORD
- DISPLAY "COMPLEXITY = ".

LOCAL DATA:
- FIRST-TIME-THRU-SW
- PRINT-AVE-N-TOTAL-SW
- PRINT-AVE-N-TOTAL-PRINT
- PRINT-SECTION-NAME-N-NUMBER
- PRINT-SECTION-NAME
- PRINT-MODULE-NAME
- PRINT-SECTION-TOTAL-N-AVERAGE

CALLS:
- GET-MODULE
- GET-WORD
- PRINT-COMPLEXITY

CHECK-MODULE-START.

PERFORM SET-PRINT-AVE-N-TOTAL-SW.

IF FIRST-TIME-THRU.

DISPLAY "COMPLEXITY = ".

LINE 1%, POSITION 20.

PERFORM PRINT-COMPLEXITY.

ELSE MOVE 1 TO FIRST-TIME-THRU.

PERFORM PRINT-COMPLEXITY.

MOVE SOURCE-WORD TO MODULE-NAME.

PERFORM GET-WORD.

IF NOT-END-OF-FILE.

IF SOURCE-WORD = "SECTION".

IF SOURCE-WORD NOT EQUAL "=".

PERFORM GET-WORD.

IF NOT-END-OF-FILE.

MOVE SOURCE-WORD TO SECTION-NUMBER.

PERFORM CHECK-FOR-AVE-N-TOTAL-PRINT.

PERFORM PRINT-SECTION-NAME-N-NUMBER.

PERFORM GET-WORD.

ELSE NEXT SENTENCE.

PERFORM CHECK-FOR-AVE-N-TOTAL-PRINT.

PERFORM PRINT-SECTION-NAME.

PERFORM GET-WORD.

ELSE PERFORM PRINT-MODULE-NAME.

SET-PRINT-AVE-N-TOTAL-SW.

IF ( NOT-FIRST-TIME-THRU AND RUNNING-SECTION-TOTAL > 0 )

PERFORM PRINT-AVE-N-TOTAL-SW.

ELSE MOVE 1 TO PRINT-AVE-N-TOTAL-SW.

IF PRINT-AVE-N-TOTAL.

PERFORM PRINT-SECTION-TOTAL-N-AVERAGE.
**3.1.3 ANALYZE COMPLEXITY**

**FUNCTION:** SCAN A CONDITIONAL, CALCULATING ITS COMPLEXITY

**INPUT** NONE

**OUTPUT** NONE

**LOCAL DATA**
- MORE-CONDITION-SW
- DONE-SW
- NO-NOT-SW
- RELATIONAL-OPERATOR-SW
- GET-MODULE-COMPLEXITY
- GET-PARENT-AND-DELETE-PARENS
- SCAN-FOR-NOT
- SCAN-QUALIFICATION-N-SUBSCRIPT
- SCAN-FOR-IS-N-NOT
- CHECK-IF-RELATIONAL-OPERATOR
- SCAN-FOR-OK-N-AND

**ANALYZE-COMPLEXITY**

MOVE Zemu TO MORE-CONDITION-Sw.
PERFORM SCAN-CONDITIONAL UNTIL NO-MORE-CONDITION.

**SCAN-CONDITIONAL**

MOVE Zemu TO DONE-Sw.
PERFORM GET-PARENT-AND-DELETE-PARENS UNTIL DONE.
MOVE Zemu TO NO-NOT-Sw.
PERFORM SCAN-FOR-NOT UNTIL NOT-NOT-Sw.
PERFORM SCAN-QUALIFICATION-N-SUBSCRIPT.
PERFORM SCAN-FOR-IS-N-NOT.
PERFORM CHECK-IF-RELATIONAL-OPERATOR.
PERFORM CHECK-OUT-CONDITION.
PERFORM SCAN-FOR-OK-N-AND.
1240*     ***************
1241*     5.1.3.1 GET PAST AND REMOVE PARENS
1242*     ***************
1243*     \* FUNCTION: SCAN PAST PARENTHESES
1244*     \* OUTPUT: SOURCE-WORD
1245*     \* LOCAL DATA: \* SUBSCRIPT-TEMP
1246*     \* CALLED BY: SCAN-CONDITIONAL
1247*     \* CALLS: GET-WORD
1248*     \*
1249*     GET-PAST-AND-REMOVE-PARENS.
1250*     IF SOURCE-WORD = "(" OR SOURCE-WORD = ")"
1251*     PERFORM GET-WORD.
1252*     ELSE MOVE 1 TO DUMB-SW.
1253*     MOVE 1 TO SUBSCRIPT-TEMP.
1254*     PERFORM SLP-PAST-PARENS.
1255*     UNTIL SOURCE-WORD ITEM (SUBSCRIPT-TEMP) NOT = "(".
1256*     UNSTRING SOURCE-WORD.
1257*     DELIMIT BY ALL SPACES OR ")".
1258*     INTO SAVE-WORD.
1259*     WITH POINTER SUBSCRIPT-TEMP.
1260*     STEP-PARENT.
1261*     ADD 1 TO SUBSCRIPT-TEMP.
FUNCTION: SCAN FOR NOT, SETTING NO-NOT-SW AS TO WHETHER ONE IS FOUND OR NOT FOUND
INPUT: SAVE-WORD
OUTPUT: NO-NOT-SW
LOCAL DATA: DONE-SW
CALLED BY: SCAN-CONDITIONAL
CALLS: GET-PAST-AND-REMOVE-PARENS
GET-WORD

SCAN-FOR-NOT
IF NOT-END-OF-FILE
    IF SAVE-WORD = "NOT"
        PERFORM GET-WORD
    MOVE ZERO TO DONE-SW
    PERFORM GET-PAST-AND-REMOVE-PARENS
    UNTIL DONE
    ELSE MOVE 1 TO NO-NOT-SW
    PERFORM GET-WORD
    ELSE MOVE 1 TO NO-NOT-SW.
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 ZER0 TO QUALIFICATION-COUNTER.

IF NOT-END-OF-ITEM
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.
1340* 
1341* 
1342* 
1343* 
1344* 
1345* 
1346* 
1347* 
1348* 
1349* 
1350* 
1351* 
1352* 
1353* 
1354*04051C  
1355*04051C  
1356* 
1357* 
1358*04052A  
1359* 
1360*  

*FUNCTION: SCAN-PAST-IS-AND-NOT IF THEY ARE FOUND.*  
*INPUTS: SOURCE-WORD*  
*OUTPUTS: SOURCE-WORD*  
*LOCAL DATA: NONE*  
*CALLED BY: SCAN-CONDITIONAL*  
*CALLED TO: GET-WORD*  

SCAN-FOR-IS-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.
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-AGRO = "OR" OR SOURCE-WORD = "AND"
PERFORM GET-WORD
ELSE MOVE 1 TO MORE-CONDITION-SW.
/ 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-VALUES

CHECK-OUT-CONDITION.

IF CLASS-CONDITION
ADD 1 TO ACCUMULATIVE-COMPLEXITY
ADD 1 TO RUNNING-SECTION-TOTAL
ADD 1 TO RUNNING-PROGRAM-TOTAL
D DISPLAY ACCUMULATIVE-COMPLEXITY
D LINE 14, POSITION 34
ELSE PERFORM LOOK-FOR-CONDITION-NAME.

LOOK-FOR-CONDITION-NAME.

MOVE ZERO TO CONDITION-FIND-FLAG.
MOVE ZERO TO SIZE-KEY.
PERFORM CO-THRU-CONDITION-NAMES
UNTIL SIZE-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,
SAFE-Mode,
LINE 21, POSITION 55,
"IN LINE NUMBER",
LINE 22, POSITION 55,
LINE-NUMBER-DISPLAY,
LINE 22, POSITION 20.

IF CONDITION-NAME-FOUND
D DISPLAY ACCUMULATIVE-COMPLEXITY
D LINE 13, POSITION 34.
**// 5.1.3.7.1 GO-THRU CONDITION NAMES
**
**FUNCTION
LOOK THROUGH THE SIRE FILE FOR AN ENTRY
THAT MATCHES THE CONDITION-NAME
CURRENTLY PRINTED TO IN THE SOURCE
PROGRAM.

**INPUT:
SIRE-KEY
SIRE-FILE
SAVE-WORD

**OUTPUT:
QUALIFICATION-COUNTER
SIRE-KEY
CONDITION-FIND-FLAG
CONDITION-NAME-ENTRY
SUBSCRIPT-TEMP
SIRE-TABLE-SUBSCRIPT
NAME-KEY

**CALLED BY:
LOOK-FOR-CONDITION-NAME

**CALLS:
NONE

**GO-THRU-CONDITION-NAME:
ADD 1 TO SIRE-KEY.
READ SIRE-FIELD 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-SIRE.
ELSE MOVE 1 TO CONDITION-FIND-FLAG.

COMPARE-QUALIFICATION-TO-SIRE.
MOVE 1 TO SUBSCRIPT-TEMP.
MOVE 1 TO SIRE-TABLE-SUBSCRIPT.
PERFORM COMPARE-SIRE UNTIL SIRE-TABLE-SUBSCRIPT = TABLE-SIZE.
OK SUBSCRIPT-TEMP > QUALIFICATION-COUNTER.
IF SUBSCRIPT-TEMP IS GREATER THAN QUALIFICATION-COUNTER
MOVE 1 TO CONDITION-FIND-FLAG.

COMPARE-SIRE.
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-TECH (SUBSCRIPT-TEMP) = SIRE-NAME
ADD 1 TO SIRE-TABLE-SUBSCRIPT.
15720  /   PRINT-SECTION-TOTAL-N-AVERAGE.  
15740  MOVE RUNNING-SECTION-TOTAL TO RUNNING-SECTION-TOTAL-OUT.  
15790  WRITE PRINT-RECORD FROM PRINT-LINE-5 AFTER ADVANCING 4 LINES  
15790  AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.  
15790  DIVIDE RUNNING-SECTION-TOTAL  
15790  BY RUNNING-SECTION-COUNT  
15790  GIVING SECTION-AVERAGE ROUNDED.  
15790  MOVE SECTION-AVERAGE TO SECTION-AVERAGE-OUT.  
15810  WRITE PRINT-RECORD FROM PRINT-LINE-6 AFTER ADVANCING 2 LINES  
15810  AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.  
15810  MOVE SECTIONS TO RUNNING-SECTION-TOTAL,  
15810  RUNNING-SECTION-COUNT.  
15840  
15850  *   PRINT-PROGRAM-TOTAL-N-AVERAGE.  
15870  PERFORM GET-TO-TOP-OF-PAGE.  
15880  MOVE RUNNING-PROGRAM-TOTAL TO RUNNING-PROGRAM-TOTAL-OUT.  
15890  WRITE PRINT-RECORD FROM PRINT-LINE-7 AFTER ADVANCING 4 LINES  
15900  AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.  
15910  DIVIDE RUNNING-PROGRAM-TOTAL  
15910  BY RUNNING-PROGRAM-COUNT  
15910  GIVING PROGRAM-AVERAGE ROUNDED.  
15940  MOVE PROGRAM-AVERAGE TO PROGRAM-AVERAGE-OUT.  
15950  WRITE PRINT-RECORD FROM PRINT-LINE-8 AFTER ADVANCING 2 LINES  
15960  AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.  
15970  *ZZZZZ END PROGRAM.  
15970  ****END OF FILE****
<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>ANS</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0004</td>
<td>490</td>
<td>0</td>
<td>FILE</td>
<td>FIXED-INPUT</td>
</tr>
<tr>
<td>0148</td>
<td>60</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>INPUT-LINE</td>
</tr>
<tr>
<td>0348</td>
<td>512</td>
<td>0</td>
<td>FILE</td>
<td>VARIABLE-INPUT</td>
</tr>
<tr>
<td>0168</td>
<td>60</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>INPUT-LINE</td>
</tr>
<tr>
<td>0230</td>
<td>132</td>
<td>0</td>
<td>FILE</td>
<td>PRINT-FILE</td>
</tr>
<tr>
<td>0188</td>
<td>132</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>PRINT-RECORD</td>
</tr>
<tr>
<td>0200</td>
<td>351</td>
<td>0</td>
<td>FILE</td>
<td>SIZE-FILE</td>
</tr>
<tr>
<td>0210</td>
<td>351</td>
<td>0</td>
<td>GRP</td>
<td>GROUP</td>
</tr>
<tr>
<td>0210</td>
<td>30</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>S-COUNT-NAME</td>
</tr>
<tr>
<td>0254</td>
<td>6</td>
<td>0</td>
<td>GRP</td>
<td>S-COUNT-NAME-POINTER</td>
</tr>
<tr>
<td>0358</td>
<td>4</td>
<td>0</td>
<td>NSU</td>
<td>S-COUNT-VALUE</td>
</tr>
<tr>
<td>0359</td>
<td>2</td>
<td>0</td>
<td>NSU</td>
<td>S-COUNT-VALUE-POINTER</td>
</tr>
<tr>
<td>0360</td>
<td>1</td>
<td>0</td>
<td>NSU</td>
<td>S-COUNT-VALUE-POINTER</td>
</tr>
<tr>
<td>0361</td>
<td>0</td>
<td>0</td>
<td>NUMERIC</td>
<td>S-ROLE-LEVEL</td>
</tr>
<tr>
<td>0120</td>
<td>510</td>
<td>0</td>
<td>FILE</td>
<td>NAME-FILE</td>
</tr>
<tr>
<td>032C</td>
<td>60</td>
<td>0</td>
<td>GRP</td>
<td>GROUP</td>
</tr>
<tr>
<td>032C</td>
<td>6</td>
<td>0</td>
<td>ANS</td>
<td>S-ROLE-NAME</td>
</tr>
<tr>
<td>033E</td>
<td>1</td>
<td>0</td>
<td>ANS</td>
<td>S-ROLE-NAME-POINTER</td>
</tr>
<tr>
<td>034F</td>
<td>65</td>
<td>0</td>
<td>ANS</td>
<td>S-ROLE-NAME-ENTRY</td>
</tr>
<tr>
<td>0410</td>
<td>8</td>
<td>0</td>
<td>NSU</td>
<td>S-ROLE-ITEM</td>
</tr>
<tr>
<td>03CB</td>
<td>80</td>
<td>0</td>
<td>GRP</td>
<td>GROUP</td>
</tr>
<tr>
<td>01C6</td>
<td>1</td>
<td>0</td>
<td>GRP</td>
<td>SOURCE-ITEM</td>
</tr>
<tr>
<td>03C6</td>
<td>1</td>
<td>0</td>
<td>ANS</td>
<td>SOURCE-QUOTE</td>
</tr>
<tr>
<td>0410</td>
<td>351</td>
<td>0</td>
<td>GRP</td>
<td>CONDITION-ENTRY</td>
</tr>
<tr>
<td>0418</td>
<td>30</td>
<td>0</td>
<td>ANS</td>
<td>CONDITION-NAME</td>
</tr>
<tr>
<td>0438</td>
<td>1</td>
<td>0</td>
<td>GRP</td>
<td>SROLE-NAME</td>
</tr>
<tr>
<td>0438</td>
<td>4</td>
<td>0</td>
<td>NSU</td>
<td>SROLE-NAME-POINTER</td>
</tr>
<tr>
<td>043A</td>
<td>2</td>
<td>0</td>
<td>NSU</td>
<td>SROLE-LEVEL</td>
</tr>
<tr>
<td>0562</td>
<td>14</td>
<td>0</td>
<td>NSU</td>
<td>SROLE-VALUE</td>
</tr>
<tr>
<td>0564</td>
<td>1</td>
<td>0</td>
<td>NSU</td>
<td>NOT-A-CONDITION-ENTRY</td>
</tr>
<tr>
<td>0575</td>
<td>2</td>
<td>0</td>
<td>NSU</td>
<td>TABLE-SIZE</td>
</tr>
<tr>
<td>0578</td>
<td>120</td>
<td>0</td>
<td>ANS</td>
<td>SOURCE-WORD</td>
</tr>
<tr>
<td>0578</td>
<td>120</td>
<td>0</td>
<td>GRP</td>
<td>CONDITION-NAME-ENTRY</td>
</tr>
<tr>
<td>0578</td>
<td>1</td>
<td>0</td>
<td>GRP</td>
<td>SOURCE-WORD-REDEFINED</td>
</tr>
<tr>
<td>0578</td>
<td>1</td>
<td>0</td>
<td>GROUP</td>
<td>SOURCE-WORD-REDEFINED</td>
</tr>
<tr>
<td>ADDR</td>
<td>SIZE</td>
<td>DEBUG</td>
<td>VALUE</td>
<td>TYPE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>0E9C</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>NAME-OP-CONDITION-NAMES</td>
</tr>
<tr>
<td>0EAE</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>NAME-KEY</td>
</tr>
<tr>
<td>0EC0</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PTR</td>
</tr>
<tr>
<td>0EE4</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>START-OF-WORD</td>
</tr>
<tr>
<td>0EFE</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>END-OF-WORD</td>
</tr>
<tr>
<td>0F0B</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>WORD-SIZE</td>
</tr>
<tr>
<td>0F1A</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>SOURCE-PTR</td>
</tr>
<tr>
<td>0F2C</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PTR-TEMP</td>
</tr>
<tr>
<td>0F36</td>
<td>2ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>DELIMITER-TEMP</td>
</tr>
<tr>
<td>0F40</td>
<td>120ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>SOURCE-WORD-TEMP</td>
</tr>
<tr>
<td>0F88</td>
<td>2ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>SOURCE-STOP</td>
</tr>
<tr>
<td>0F88</td>
<td>2ANS</td>
<td>0</td>
<td>NUMERIC</td>
<td>ID-1</td>
</tr>
<tr>
<td>0F8C</td>
<td>2ANS</td>
<td>0</td>
<td>NUMERIC</td>
<td>SOURCE-TABLE-SUBSCRIPT</td>
</tr>
<tr>
<td>0F9E</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>SUBSCRIPT-TEMP</td>
</tr>
<tr>
<td>0FDD</td>
<td>2ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>CURRENT-LEVEL</td>
</tr>
<tr>
<td>0FD2</td>
<td>2ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>LAST-LEVEL</td>
</tr>
<tr>
<td>0FDA</td>
<td>18NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>RUNNING-SECTION-COUNT</td>
</tr>
<tr>
<td>0FEB</td>
<td>18NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>RUNNING-PROGRAM-COUNT</td>
</tr>
<tr>
<td>0FF8</td>
<td>18NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>RUNNING-SECTION-TOTAL</td>
</tr>
<tr>
<td>100A</td>
<td>18NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>RUNNING-PROGRAM-TOTAL</td>
</tr>
<tr>
<td>101C</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>SECTION-AVERAGE</td>
</tr>
<tr>
<td>102E</td>
<td>10NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PROGRAM-AVERAGE</td>
</tr>
<tr>
<td>1040</td>
<td>30ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>CONDITION-NAME</td>
</tr>
<tr>
<td>105E</td>
<td>1NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>CONDITION-NAME</td>
</tr>
<tr>
<td>105F</td>
<td>1NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>CONDITION-NAME</td>
</tr>
<tr>
<td>ADDR</td>
<td>SIZE</td>
<td>DEBUG</td>
<td>ORIGIN</td>
<td>TYPE</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>1060</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1061</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1062</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1063</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1064</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1065</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1066</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1067</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1068</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1069</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>106A</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>106B</td>
<td>4</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>106C</td>
<td>4</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC EDITED</td>
</tr>
<tr>
<td>1073</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
<tr>
<td>1074</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC CONDITION-NAME</td>
</tr>
</tbody>
</table>

**Requied Runtime Elements:**
- NUCLEUS
- ACCEPT/DISPLAY
- LINEAGE
- MULTIPLY/DIVIDE
- STRING

**Local Byte Size:** 6702

**Global Byte Size:** 1086

**Total Byte Size:** 7788
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.
IDENTIFICATION DIVISION.

PROJECT-NAME: HALSTEAD'S-MEASURE.

AUTHOR: ANDRE W. MEALS.

DATE-WRITTEN: 08/07/14.

DATE-COMPiled: 08/03/14.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-PROGRAM: HALSTEAD'S METRICS EVALUATION TOOL.
1.*)  
20.*)  
21.*)  
22.*)  
FILE-CONTROL.  
23.*)  
24.*)  
SELECT FIXED-INPUT ASSIGN TO DISC.  
25.*)  
SELECT VARIABLE-INPUT ASSIGN TO DISC.  
26.*)  
SELECT PHYS-FILE ASSIGN TO PRINTER.  
27.*)  
SELECT OPERAND-FILE ASSIGN TO DISC.  
28.*)  
ORGANIZATION IS RELATIVE;  
29.*)  
RELATIVE KEY IS OPERAND-KEY.  
30.*)  
SELECT LINK-FILE ASSIGN TO DISC  
31.*)  
ORGANIZATION IS RELATIVE;  
32.*)  
ACCESS MODE IS RANDOM;  
33.*)  
RELATIVE KEY IS LINK-KEY.  
34.*)  
I-O-CONTROL.  
35.*)  
SAME RECORD AREA FOR FIXED-INPUT, VARIABLE-INPUT.
DATA DIVISION.
FD FILE-INPUT.
   BLOCK CONTAINS 6 RECORDS
   RECORD CONTAINS 80 CHARACTERS
   LABEL RECORDS ARE STANDARD
   VALUE OF FILE-ID IS "I".
   01 INPUT-LINE PIC X(40).
   * 
   FD VARIABLE-INPUT.
   BLOCK CONTAINS 512 CHARACTERS
   RECORD VARYING 6 TO 80 CHARACTERS DEPENDING ON ID-1
   LABEL RECORDS ARE STANDARD
   VALUE OF FILE-ID IS "I".
   01 INPUT-LINE PIC X(80).
   * 
   FD PRINT-FILE.
   BLOCK CONTAINS 1 RECORDS
   RECORD CONTAINS 122 CHARACTERS
   LABEL RECORDS ARE OMITTED
   LINES AT TOP 3.
   LINES AT BOTTOM 5.
   01 PRINT-RECORD PIC X(132).
   * 
   FD OPERAND-FILE.
   BLOCK CONTAINS 3 RECORDS
   LABEL RECORDS ARE STANDARD
   VALUE OF FILE-ID IS "OPERAND".
   01 OPERAND-RECORD.
      05 A-OPERAND-REFERENCE PIC X(122).
      05 OPERAND-REFERENCE-COUNTER PIC 9(16).
   * 
   FD LINK-FILE.
   BLOCK CONTAINS 14 RECORDS
   LABEL RECORDS ARE STANDARD
   VALUE OF FILE-ID IS "LINK".
   01 OPERAND-LINK-RECORD.
      05 OPERAND-POINTER PIC 9(16).
      05 FORWARD-Link PIC 9(16).
DATA DIVISION.
FILE SECTION.
FD FILE-INPUT
  BLOCK CONTAINS 6 RECORDS
  RECORD CONTAINS 80 CHARACTERS
  LABEL RECORDS ARE STANDARD
  VALUE OF FILE-ID IS "1".
  01 INPUT-LINE PIC X(60).

FD VARIABLE-INPUT
  BLOCK CONTAINS 512 CHARACTERS
  RECORD VARYING 6 TO 80 CHARACTERS DEPENDING ON ID-1
  LABEL RECORDS ARE STANDARD
  VALUE OF FILE-ID IS "1".
  01 INPUT-LINE PIC X(60).

FD PRINT-FIELD
  BLOCK CONTAINS 1 RECORDS
  RECORD CONTAINS 132 CHARACTERS
  LABEL RECORDS ARE OMITTED
  LINEAGE IS 74 LINES.
  LINES AT TOP 3.
  LINES AT BOTTOM 5.
  01 PRINT-RECORD PIC X(132).

FD OPERAND-FILE
  BLOCK CONTAINS 3 RECORDS
  LABEL RECORDS ARE STANDARD
  VALUE OF FILE-ID IS " OPERAND".
  01 OPERAND-RECORD.
    05 A-OPERAND-NAME PIC X(122).
    05 OPERAND-REFERENCE-COUNTER PIC 9(10).

FD LINK-FILE
  BLOCK CONTAINS 14 RECORDS
  LABEL RECORDS ARE STANDARD
  VALUE OF FILE-ID IS " LINK".
  01 OPERAND-REFERENCE-RECORD.
    05 OPERAND-REFERENCE-VALUE PIC 9(10).
* WORKING-STORAGE SECTION. *
70*
* 01 SOURCE-LINE.
71* 95 SEQUENCE-NUMBER PIC X(6).
72* 05 LINES-OF-TEXT PIC X(6).
73* 96 COMMENT-OR-DEBUG VALUES ARE "*", "D", "/".
74* 88 LINES-CONTINUATION VALUE IS "-".
75* 05 TEXT-LINE PIC X(65).
76* 05 ID-FIELD PIC X(6).
77* 01 SOURCE-LINE-REDEFINED REDEFINES SOURCE-LINE.
78* 05 SOURCE-ITEMS OCCURS 80 TIMES.
79* 10 SOURCE-LINE-ITEM PIC X.
80*
* 01 SOURCE-WORD PIC X(122).
81* 01 SOURCE-WORD-REDEFINED REDEFINES SOURCE-WORD.
82* 05 SOURCE-WORDS OCCURS 122 TIMES.
83* 10 SOURCE-WORD-ITEM PIC X.
84* *
* 01 SAVE-WORD PIC X(122).
85* 01 SAVE-WORD-REDEFINED REDEFINES SAVE-WORD.
86* 05 SAVE-WORDS OCCURS 122 TIMES.
87* 10 SAVE-WORD-ITEM PIC X.
88*
* 01 OPERATOR-ENTRIES.
89* 05 OPERATOR-TABLE OCCURS 90 TIMES.
90* 10 OPERATOR-ENTRY PIC X(120).
91* 10 OPERATOR-COUNTER PIC 9(16).
92* *
* 01 ALPHABET-ENTRIES.
93* 05 ALPHABET-TABLE OCCURS 26 TIMES.
94* 10 ALPHABET-ENTRY PIC X.
95* 10 ALPHABET-LINK PIC 99.
96* *
* 01 OPERAND-HEAD-LINKS.
97* 05 HEAD-LINK-LIST OCCURS 27 TIMES.
98* 10 HEAD-LINK PIC 9(18).
99* *
* 01 OPERAND-ENTRY.
100* 05 OPERAND-NAME PIC X(122).
101* 05 OPERAND-COUNTER PIC 9(18).
102*
* 01 THE-TIME.
103* 05 MINS PIC 99.
104* 05 SECONDS PIC 99.
180* 01 PRINT-LINE-4.
181* 05 FILLER PIC X(23) VALUE IS SPACES.
182* 05 FILLER PIC X(10).
183* 05 FILLER PIC X(4).
184* 05 FILLER PIC X(2) VALUE IS "TOTAL FREQUENCY OF OPERATORS".
185* 05 FILLER PIC X(5) VALUE IS SPACES.
186* 05 FILLER PIC X(2) VALUE IS "FL".
187* 05 FILLER PIC X(33) VALUE IS SPACES.
188* 05 OUTPUT-VALUE=4 PIC Z(17)99.
189* 05 PRINT-LINE-5.
190* 05 FILLER PIC X(26) VALUE IS SPACES.
191* 05 FILLER PIC X(27).
192* 05 FILLER PIC X(10).
193* 05 FILLER PIC X(17) VALUE IS "TOTAL FREQUENCY OF OPERANDS".
194* 05 FILLER PIC X(18) VALUE IS SPACES.
195* 05 FILLER PIC X(2) VALUE IS "F2".
196* 05 FILLER PIC X(33) VALUE IS SPACES.
197* 05 OUTPUT-VALUE=5 PIC Z(17)99.
198* 05 PRINT-LINE-6.
199* 05 FILLER PIC X(53) VALUE IS SPACES.
200* 05 FILLER PIC X(19).
201* 05 FILLER PIC X(18).
202* 05 FILLER PIC X(51) VALUE IS "COMPUTED MEASURES".
203* 05 FILLER PIC X(5) VALUE IS SPACES.
204* 05 FILLER PIC X(18).
205* 05 FILLER PIC X(17) VALUE IS "PROGRAM VOCABULARY".
206* 05 FILLER PIC X(59) VALUE IS SPACES.
207* 05 FILLER PIC X(5) VALUE IS "N1+N2".
208* 05 FILLER PIC X(33) VALUE IS SPACES.
209* 05 OUTPUT-VALUE=6 PIC Z(17)99.
210* 01 PRINT-LINE-7.
211* 05 FILLER PIC X(28) VALUE IS SPACES.
212* 05 FILLER PIC X(14).
213* 05 FILLER PIC X(21) VALUE IS "PROGRAM LENGTH".
214* 05 FILLER PIC X(21) VALUE IS SPACES.
215* 05 FILLER PIC X(2) VALUE IS "F1+F2".
216* 05 FILLER PIC X(10) VALUE IS SPACES.
217* 05 OUTPUT-VALUE=7 PIC Z(17)99.
218* 01 PRINT-LINE-8.
219* 05 FILLER PIC X(28) VALUE IS SPACES.
220* 05 FILLER PIC X(21).
221* 05 FILLER PIC X(53) VALUE IS "ESTIMATED PROGRAM LENGTH".
222* 05 FILLER PIC X(11) VALUE IS SPACES.
223* 05 FILLER PIC X(11).
224* 05 FILLER PIC X(130).
225* 05 FILLER PIC X(5). VALUE IS "N=F1LOG2(N1)+N2LOG2(N2)".
226* 05 FILLER PIC X(14). VALUE IS SPACES.
227* 05 OUTPUT-VALUE=8 PIC Z(17)99.
01 PRINT-LINE-9.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(119).
234 "PROGRAM VOLUME".
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS "Y = (F1+F2)LOG2(1+X)*N1".
05 FILLER PIC X(28) VALUE IS SPACES.
05 OUTPUT-VALUE-9 PIC 9(9.99). 

01 PRINT-LINE-10.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
244 "IMPLEMENTATION LEVEL".
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS "L = APPR. Z2\{X2\}/((X1)(F2))".
05 FILLER PIC X(28) VALUE IS SPACES.
05 OUTPUT-VALUE-10 PIC 9(9.99). 

01 PRINT-LINE-11.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
254 "POTENTIAL VOLUME".
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS "V* = L/(L\{V\})".
05 FILLER PIC X(28) VALUE IS SPACES.
05 OUTPUT-VALUE-11 PIC 9(9.99). 

01 PRINT-LINE-12.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
264 "EFFORT METRIC".
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS "G = V*\{L\}".
05 FILLER PIC X(28) VALUE IS SPACES.
05 OUTPUT-VALUE-12 PIC 9(9.99). 

01 PRINT-LINE-13.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
274 "PROGRAMMING TIME".
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS "T = CG/18".
05 FILLER PIC X(28) VALUE IS SPACES.
05 OUTPUT-VALUE-13 PIC 9(9.99). 

01 PRINT-LINE-14.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
284 "SEC.".
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
05 OUTPUT-VALUE-14 PIC 9(9.99). 

01 PRINT-LINE-15.
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
294 "SEC.".
05 FILLER PIC X(28) VALUE IS SPACES.
05 FILLER PIC X(28) VALUE IS SPACES.
05 OUTPUT-VALUE-15 PIC 9(9.99).
01 PRINT-LINE-14.
  00 FILLER PIC X(107) VALUE IS SPACES.
  00 OUTPUT-VALUE-14 PIC Z(99).9(4).
  00 FILLER PIC X VALUE IS SPACES.
  00 FILLER PIC X(4) VALUE IS "MIN.

01 PRINT-LINE-15.
  00 FILLER PIC X(107) VALUE IS SPACES.
  00 OUTPUT-VALUE-15 PIC Z(99).9(4).
  00 FILLER PIC X VALUE IS SPACES.
  00 FILLER PIC X(4) VALUE IS "HRS.

01 PRINT-LINE-16.
  00 FILLER PIC X(280) VALUE IS SPACES.
  00 FILLER PIC X(144) VALUE IS "LANGUAGE LEVEL".
  00 FILLER PIC X(12) VALUE IS SPACES.
  00 FILLER PIC X(9) VALUE IS "FILELEV".
  00 FILLER PIC X(135) VALUE IS SPACES.
  00 OUTPUT-VALUE-16 PIC Z(191).9(4).

01 PRINT-LINE-17.
  00 FILLER PIC X(293) VALUE IS "FREQUENCY OF OPERATORS IN".
  00 FILLER PIC X(138) VALUE IS "DESCENDING ORDER".

01 PRINT-LINE-18.
  00 OPERATOR-OUTPUT-COUNTER PIC Z(4).
  00 FILLER PIC X(15) VALUE IS SPACES.
  00 OPERATOR-OUTPUT-ELEMENT PIC X(301).

01 PRINT-LINE-19.
  00 FILLER PIC X(24) VALUE IS "FREQUENCY OF OPERANDS IN".
  00 FILLER PIC X(138) VALUE IS "DESCENDING ORDER".

01 PRINT-LINE-20.
  00 OPERAND-OUTPUT-COUNTER PIC Z(4).
  00 FILLER PIC X(4) VALUE IS SPACES.
  00 OPERAND-OUTPUT-ELEMENT PIC X(122).
THE-DATE  PIC 9(6).
PAGE-COUNT  PIC 9(3) VALUE IS ZERO.
PTR  PIC 9(10) VALUE IS 75.
START-OF-WORD  PIC 9(10) VALUE IS ZERO.
END-OF-WORD  PIC 9(10).
WORD-SIZE  PIC 9(10).
SOURCE-PTR  PIC 9(10).
PTR-TEMP  PIC 9(10).
SUBSCRIPT-TEMP  PIC 9(10).
DELIMITER-TEMP  PIC XX.
SOURCE-WORD-TEMP  PIC X(122).
SOURCE-STOP  PIC XY.
SAVE-STOP  PIC XX.
ID-1  PIC 99 VALUE IS 80.
LINE-NUMBER  PIC 9(14) VALUE IS ZERO.
LINE-NUMBER-DISPLAY  PIC 2(14).
OPAND-KEY  PIC 9(10).
LINK-KEY  PIC 9(10).
NUMHEX-OF-LINKS  PIC 9(10) VALUE IS ZERO.
LAST-LINK  PIC 9(10).
LINK-INDEX  PIC 9(10).
ITEM-REING-CONSIDERED  PIC 9(10).
ONE-TO-PRINT  PIC 9(10).
LAST-ITEM-PRINTED  PIC 9(10).
CURRENT-HIGH  PIC 9(10).
LAST-HIGH  PIC 9(10).
NUMBER-OF-ITEMS-PRINTED  PIC 9(10).
FILE-FLAG                   PIC 9         VALUE IS ZERO.
60 FIXED-LENGTH-FILE       VALUE IS ZERO.
43 VARIABLE-LENGTH-FILE    VALUE IS 1.
FILE-FLAG                   PIC 9         VALUE IS ZERO.
40 NOT-END-OF-FILE         VALUE IS 1.
60 END-OF-FILE             VALUE IS 1.
77 AREA-A-FLAG             PIC 9         VALUE IS ZERO.
60 AREA-A-FLAG             VALUE IS 1.
CONSUME-SW                  PIC 9         VALUE IS ZERO.
60 CONSUE-DONE             VALUE IS 1.
QUOTE-SW                    PIC 9         VALUE IS ZERO.
60 BALANCED-QUOTE          VALUE IS 1.
JUNE-SW                    PIC 9         VALUE IS ZERO.
60 DONE                    VALUE IS 1.
77 OPERAND-FIND-FLAG       PIC 9         VALUE IS ZERO.
60 OPERAND-NOT-FOUND       VALUE IS 1.
60 OPERAND-FOUND
PROCEDURE DIVISION.

DECLARATIVES.

OPEN-ERROR SECTION.

USE AFTER STANDARD EXCEPTION PROCEDURE ON FIXED-INPUT.

OPEN-VARIABLE.

OPEN INPUT VARIABLE-INPUT.

MOVE 1 TO FILE-FLAG.

OPEN-END.

END DECLARATIVES.
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 OPERATOR 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
  7.1.1 GROUP OPERATOR TABLE
  7.1.2 CALCULATE PROGRAM LENGTH
  7.1.3 CALCULATE ESTIMATE OF LENGTH

7.2 CALCULATE PROGRAM ELEMENTS

7.3 CALCULATE PROGRAM VOLUME
  7.3.1 COMPUTE LOG BASE TWO

7.4 CALCULATE IMPLEMENTATION LEVEL

7.5 CALCULATE FUNCTION LEVEL

7.6 CALCULATE EFFORT METRIC

7.7 CALCULATE PROGRAMMING TIME

7.8 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
FUNCTION: OUTERMOST ROUTINE (DRIVER)

INPUT:  NONE

OUTPUT: NONE

LOCAL DATA: FIXED-LENGTH-FILE

CALLED BY: GET-TIME-AND-DATE

CALLS: INITIALIZE-OPERATOR-TABLE

INITIALIZE-ALPHABET-TABLE

PROCESS-PROGRAM-ID-PARAGRAPH

PROCESS-PROCEDURE-DIVISION

COMPUTE-MEASURES

OUTPUT-RESULTS

MAIN-PROGRAM SECTION.

MAIN:

PERFORM GET-TIME-AND-DATE.

OPEN INPUT FIXED-INPUT.

OPEN OUTPUT PRINT-FILE.

UPLO INPUT OPERAND-FILE.

UPLO OUTPUT LINK-FILE.

CLOSE OPERAND-FILE.

CLOSE LINK-FILE.

UPLO I-O UPRAND-FILE.

UPLO I-O LINK-FILE.

PERFORM INITIALIZE-OPERATOR-TABLE.

PERFORM INITIALIZE-ALPHABET-TABLE.

PERFORM PROCESS-PROGRAM-ID-PARAGRAPH.

PERFORM PROCESS-PROCEDURE-DIVISION.

PERFORM COMPUTE-MEASURES.

PERFORM OUTPUT-RESULTS.

IF FIXED-LENGTH-FILE

CLOSE FIXED-INPUT.

ELSE CLOSE VARIABLE-INPUT.

CLOSE PRINT-FILE.

CLOSE OPERAND-FILE.

CLOSE LINK-FILE.

STOP RUN.
FUNCTION  ISOLATE THE NEXT TOKEN IN THE COMDL source

INPUTS  PTR
OUTPUTS  SOURCE-WORD
LOCAL DATA  NONE
CALLED BY  PROCESS-PROGRAM-ID-PARAGRAPH
PROCEDURE-PROCEDURE-DIVISION
COLLECT-OPERATORS-AND-OPERANDS
GET-PAST-AND-REMOVE-PARENTS
CALLS  READ-NEXT
ISOLATE-WORD

**COMMON-SUBROUTINE SECTION**

**FUNCTION**  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

PERFORM GET-WORD-ACTUAL.
IF PTR GREATER THAN 72
PERFORM READ-NEXT
UNTIL NOT END-OF-FILE
OR END-OF-FILE
PERFORM ISOLATE-WORD.
FUNCTION: INPUT THE NEXT CHARACTER OF THE COBOL
SOURCE FILE

INPUT: NONE
OUTPUT: PTR
LOCAL DATA: LINE-NUMBER
CALLED BY: GET-WORD-ACTUAL
CONSUME-LITERAL-CHARACTER
ABSORB-SPACES
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 8 TO PTR.
MOVL 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.
**Q1.2 ISOLATE WORD ROUTINE**

**FUNCTION:** ISOLATE A TOKEN IN THE COBOL SOURCE PROGRAM

**INPUT:**
- PTR
- SOURCE-LINE

**OUTPUT:**
- SOURCE-WORD
- AREA-A-FLAG
- SOURCE-STOP
- WORD-SIZE
- PTR

**CALLED BY:** GET-WORD-ACTUAL

**CALLS:** CONSUM-LITERAL, EXAMINE-AND-ISOLATE-SUBSCRIPTS

---

**ISOLATE-WORD**

MOVE SPACES TO SOURCE-WORD.

MOVE PTR TO START-OF-WORD.

ISOLATE-WORD:

IF SOURCE-LINE-1ST-M (PTR) = "***"

PLATFORM CONSUM-LITERAL

ELSE UNSTRING SOURCE-LINE

DELIMITED BY "", " " OR "" OR "$" OR ALL SPACES INTO SOURCE-WORD.

DELIMIT IN SOURCE-STOP.

COUNT IN WORD-SIZE.

WITH PRTIR PTR

PERFORM EXAMINE-AND-ISOLATE-SUBSCRIPTS.

IF START-OF-WORD LESS THAN 12

AND SOURCE-WORD-ITEM1 NOT EQUAL "***"

AND SOURCE-WORD NOT EQUAL "" OR "$"

MOVE 1 TO AREA-A-FLAG.

ELSE MOVE ZERO TO AREA-A-FLAG.

DISPLAY SPACES, LINE 7, SIZE 60.

DISPLAY "***".

LINE 7, POSITION PTR.

DISPLAY "*".

SOURCE-WORD.

DISPLAY ".".

LINE 7, POSITION ZZ.
FUNCTION: ISOLATE A LITERAL TOKEN IN THE COBOL TEXT

INPUT: PTR

OUTPUT: SOURCE-WORD

LOCAL DATA:
QUOTE-SW
CONSUME-SW
SOURCE-PTR
CALLS READ-NEXT

******************************************************************************
657* CONSUME-LITERAL.
658* MOVE ZERO TO SOURCE-PTR.
659* MOVE ZERO TO CONSUME-SW.
660* MOVE 1 TO QUOTE-SW.
661* PERFORM CONSUME-LITERAL-CHARACTER.
662* PERFORM ABSORB-SPACES.
663* END-OF-Main.
664* CONSUME-LITERAL-CHARACTERS.
665* ADD 1 TO SOURCE-PTR.
666* IF PTR GREATER THAN 72.
667* PERFORM READ-NEXT.
668* PERFORM ABSORB-SPACES.
669* IF BALANCED-QUOTE.
670* IF SOURCE-WORD-ITEM (PTR) NOT EQUAL "".
671* MOVE 1 TO CONSUME-SW.
672* MOVE SOURCE-WORD-ITEM (PTR) TO SOURCE-STOP.
673* ELSE PERFORM EAT-LITERAL-CHARACTER.
674* ELSE PERFORM EAT-LITERAL-CHARACTER.
675* EAT-LITERAL-CHARACTER.
676* MOVE SOURCE-WORD-ITEM (PTR) TO SOURCE-WORD-ITEM (SOURCE-PTR).
677* END-OF-Main.
678* END-OF-Main.
679* END-OF-Main.
680* END-OF-Main.
681* IF PTR GREATER THAN 72.
682* PERFORM READ-NEXT.
683* PERFORM ABSORB-SPACES.
684* ADD 1 TO PTR.
685* END-OF-Main.
686* END-OF-Main.
687* END-OF-Main.
688* END-OF-Main.
689* END-OF-Main.
690* END-OF-Main.
691* END-OF-Main.
692* END-OF-Main.
693* END-OF-Main.
694* END-OF-Main.
FUNCTION:  STRIP OFF AND EXAMINE SUBSCRIPTS

** Local Data: 
- END-OF-WORD
- SOURCE-WORD-TEMP
- DELIMITER-TEMP
- PTR-TEMP

** Calls: 
- NONE

** 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 "\n  AND SOURCE-WORD NOT EQUAL "\n  MOVE 1 TO PTR-TEMP
  UNSTRING SOURCE-WORD
  DELIMITED BY "\" OR SPACES
  INTO SOURCE-WORD-TEMP
  DELIMITER IN UDELIMITER-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.
FUNCTION: SLEW THE OUTPUT LISTING TO THE TOP OF THE PAGE AND PRINT THE HEADING WITH PAGE NUMBER.

INPUT: PAGE-COUNT

OUTPUT: PAGE-COUNT

LOCAL DATA: PRINT-RECORD

CALLED BY: OUTPUT-OPERATOR-FREQUENCIES

OUTPUT-AN-OPERATOR

OUTPUT-AN-OPERAND

CALLS: NONE

---

GET-TO-TOP-OF-PAGE:
ADD 1 TO PAGE-COUNT.
MOVE PAGE-COUNT TO PAGE-NUMBER.
WRITE PRINT-RECORD FROM HEADER AFTER ADVANCING PAGE.
FUNCTION: OUTPUT A BLANK LINE TO THE OUTPUT LISTING

INPUT: NONE

OUTPUT: PRINT-RECORD

LOCAL DATA: NONE

CALLED BY: OUTPUT-AN-OPERATOR

PRINT-RECORD AFTER ADVANCING 1 LINES.
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

*-----------------------------------------------------*
821: INITIAL-OPERATOR-TABLE
822: INITIAL-OPERATOR-TABLE
823: INITIAL-OPERATOR-TABLE
824: INITIAL-OPERATOR-TABLE
825: INITIAL-OPERATOR-TABLE
826: INITIAL-OPERATOR-TABLE
827: INITIAL-OPERATOR-TABLE
828: INITIAL-OPERATOR-TABLE
829: INITIAL-OPERATOR-TABLE
830: INITIAL-OPERATOR-TABLE
831: INITIAL-OPERATOR-TABLE
832: INITIAL-OPERATOR-TABLE
833: INITIAL-OPERATOR-TABLE
834: INITIAL-OPERATOR-TABLE
835: INITIAL-OPERATOR-TABLE
836: INITIAL-OPERATOR-TABLE
837: INITIAL-OPERATOR-TABLE
838: INITIAL-OPERATOR-TABLE
839: INITIAL-OPERATOR-TABLE
840: INITIAL-OPERATOR-TABLE
841: INITIAL-OPERATOR-TABLE
842: INITIAL-OPERATOR-TABLE
843: INITIAL-OPERATOR-TABLE
844: INITIAL-OPERATOR-TABLE
845: INITIAL-OPERATOR-TABLE
846: INITIAL-OPERATOR-TABLE
847: INITIAL-OPERATOR-TABLE
848: INITIAL-OPERATOR-TABLE
849: INITIAL-OPERATOR-TABLE
850: INITIAL-OPERATOR-TABLE
851: INITIAL-OPERATOR-TABLE
852: INITIAL-OPERATOR-TABLE
853: INITIAL-OPERATOR-TABLE
854: INITIAL-OPERATOR-TABLE
855: INITIAL-OPERATOR-TABLE
856: INITIAL-OPERATOR-TABLE
857: INITIAL-OPERATOR-TABLE
858: INITIAL-OPERATOR-TABLE
859: INITIAL-OPERATOR-TABLE
860: INITIAL-OPERATOR-TABLE
861: INITIAL-OPERATOR-TABLE
862: INITIAL-OPERATOR-TABLE
863: INITIAL-OPERATOR-TABLE
864: INITIAL-OPERATOR-TABLE
865: INITIAL-OPERATOR-TABLE

*-----------------------------------------------------*
MOVE "DISABLE"
TO OPERATOR-ELEMENT (20).

MOVE "DISPLAY"
TO OPERATOR-ELEMENT (21).

MOVE "DIVIDE"
TO OPERATOR-ELEMENT (22).

MOVE "DOWN"
TO OPERATOR-ELEMENT (23).

MOVE "ELSE"
TO OPERATOR-ELEMENT (24).

MOVE "ENABLE"
TO OPERATOR-ELEMENT (25).

MOVE "END"
TO OPERATOR-ELEMENT (26).

MOVE "END-OF-PAGE"
TO OPERATOR-ELEMENT (27).

MOVE "EQUAL"
TO OPERATOR-ELEMENT (28).

MOVE "GIVING"
TO OPERATOR-ELEMENT (29).

MOVE "GO"
TO OPERATOR-ELEMENT (30).

MOVE "GREATER"
TO OPERATOR-ELEMENT (31).

MOVE "IF"
TO OPERATOR-ELEMENT (32).

MOVE "IN"
TO OPERATOR-ELEMENT (33).

MOVE "INITIATE"
TO OPERATOR-ELEMENT (34).

MOVE "INSPECT"
TO OPERATOR-ELEMENT (35).

MOVE "INTO"
TO OPERATOR-ELEMENT (36).

MOVE "INVALID"
TO OPERATOR-ELEMENT (37).

MOVE "LESS"
TO OPERATOR-ELEMENT (38).

MOVE "LIKE"
TO OPERATOR-ELEMENT (39).

MOVE "MARCH"
TO OPERATOR-ELEMENT (40).

MOVE "MOVE"
TO OPERATOR-ELEMENT (41).

MOVE "MULTIPLY"
TO OPERATOR-ELEMENT (42).

MOVE "NEXT"
TO OPERATOR-ELEMENT (43).

MOVE "NOT"
TO OPERATOR-ELEMENT (44).

MOVE "OPEN"
TO OPERATOR-ELEMENT (45).

MOVE "OPEN"
TO OPERATOR-ELEMENT (46).

TO OPERATOR-ELEMENT (47).
978+2046D  MOVE "MP" TO OPERATOR-ELEMENT (761).
979#  MOVE "MPIN" TO OPERATOR-ELEMENT (771).
980#  MOVE "MAYING" TO OPERATOR-ELEMENT (781).
981#  MOVE "M" TO OPERATOR-ELEMENT (791).
982#  MOVE "M" TO OPERATOR-ELEMENT (801).
983#  MOVE "M" TO OPERATOR-ELEMENT (811).
984#  MOVE "M" TO OPERATOR-ELEMENT (821).
985#  MOVE "M" TO OPERATOR-ELEMENT (831).
986#  MOVE "M" TO OPERATOR-ELEMENT (841).
987#  MOVE "M" TO OPERATOR-ELEMENT (853).
988#  MOVE "M" TO OPERATOR-ELEMENT (863).
989#  MOVE "M" TO OPERATOR-ELEMENT (873).
990#  MOVE "M" TO OPERATOR-ELEMENT (883).
991#  MOVE "M" TO OPERATOR-ELEMENT (893).
992#  MOVE "M" TO OPERATOR-ELEMENT (903).
993#  MOVE "M" TO OPERATOR-ELEMENT (913).
994#  MOVE "M" TO OPERATOR-ELEMENT (923).
995#  MOVE "M" TO OPERATOR-ELEMENT (933).
996#  MOVE "M" TO OPERATOR-ELEMENT (943).
997#  MOVE "M" TO OPERATOR-ELEMENT (953).
998#  MOVE "M" TO OPERATOR-ELEMENT (963).
/* initialize alphabet table routine */

FUNCTION: initialize the entries in the alphabet table and in the head-link-list

INPUT: NONE

OUTPUT: ALPHABET-ENTRIES

OPERAND: HEAD-LEGS

LOCAL DATA: NONE

CALLED BY: MAIN

CALLS: NONE

*******************************************************************************/

initialize-alphabet-table:

move zeroes to operand-head-links

move 'A' to alphabet-item

move 'B' to alphabet-item

move 'C' to alphabet-item

move 'D' to alphabet-item

move 'E' to alphabet-item

move 'F' to alphabet-item

move 'G' to alphabet-item

move 'H' to alphabet-item

move 'I' to alphabet-item

move 'J' to alphabet-item

move 'K' to alphabet-item

move 'L' to alphabet-item

move 'M' to alphabet-item

move 'N' to alphabet-item

move 'O' to alphabet-item

move 'P' to alphabet-item

move 'Q' to alphabet-item

move 'R' to alphabet-item

move 'S' to alphabet-item

move 'T' to alphabet-item

move 'U' to alphabet-item

move 'V' to alphabet-item

move 'W' to alphabet-item

move 'X' to alphabet-item

move 'Y' to alphabet-item

move 'Z' to alphabet-item

move alphabet-item to alphabet-table
11044 /*
11045  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*
11046 *  S  PROCESS PROGRAM ID PARAGRAPH ROUTINE                                *
11047 *                                                                            *
11048 *  FUNCTION: SCAN FOR PROGRAM-ID TO PUT IN THE LISTING                    *
11049 *  INPUT:  SOURCE-WORD                                                     *
11050 *  OUTPUT: PAGE-COUNT                                                      *
11051 *  SOURCE-WORD                                                             *
11052 *  LOCAL DATA: NONE                                                       *
11053 *  CALLED BY: MAIN                                                        *
11054 *  CALLS: NONE                                                            *
11055 *                                                                            *
11056 */
11057  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*
11058  +
11059 #3020A                PROGRAM-IDENTIFICATION SECTION 52.                    *
11060 *
11061 #3020A                  PROGRAM-ID-PARAGRAPH.                             *
11062 #3020A                  DISPLAY "HALSTEAD'S COMPLEXITY MEASURE":
11063  D LINE 1: POSITION 21; ERASE;
11064  D "SOURCE WORD = " LINE 9: POSITION 5;
11065 #3020A                  PERFORM GET-WORD.
11066  UNTIL SOURCE-WORD = "PROGRAM-ID":
11067  D PERFORM GET-WORD.
11068 #3020A                  MOVE SOURCE-WORD TO PROGRAM-NAME.
11069 #3020A                  MOVE 1 TO PAGE-COUNT.
11070  D DISPLAY "PROGRAM NAME = "
11071  D LINE 3: POSITION 5;
11072  D PROGRAM-NAME;
11073  D LINE 3: POSITION 20;
11074 #3020A                  MOVE PAGE-COUNT TO PAGE-NUMBER.
11075 #3020F2                 WRITE PRINT-RECORD FROM HEADER.  */
1126  PROCEDURE DIVISION.
1127  PERFORM GET-WORD.
1128  UNTIL SOURCE-WORD = "PROCEDURE".
1129  PERFORM GET-WORD.
1130  PERFORM COLLECT-OPERATORS-AND-OPERANDS UNTIL END-OF-FILE.
1131  COLLECT-OPERATORS-AND-OPERANDS.
1132  PERFORM GET-WORD.
1133  PERFORM GET-WORD.
1134  UNTIL END-OF-FILE.
1135  OR SOURCE-WORD NOT = "SECTION" AND NOT AREA-A-OCCLUDED.
1136  IF NOT END-OF-FILE.
1137  MOVE ZERO TO DONE-SW.
1138  PERFORM GET-FAST-AND-REMOVE-PARENS.
1139  UNTIL DONE.
1140  SET OPERATOR-INDEX TO 1.
1141  SEARCH OPERATIONS-TABLE VARYING OPERATOR-INDEX.
1142  AT END PERFORM REGISTER-OPERAND.
1143  WHEN OPERATOR-CLEMENT(OPERATOR-INDEX) = SAVE-WORD.
1144  PERFORM REGISTER-OPERATOR.
*** GET-PAST-AND-REMOVE-PARENTS Routines ***

FUNCTION: SCAN FOR ANY PARENTHESES AND REMOVE THEM

INPUTS: SOURCE-WORD

OUTPUT: SOURCE-WORD

CALL BY: COLLECT-OPERATORS-AND-OPERANDS

* 04 0433E GET-PAST-AND-REMOVE-PARENTS
  01 04033E IF SOURCE-WORD = "("
    01 04033E PERFORM REGISTER-LEFT-PAREN
  01 04033E PERFORM GET-WORD
  01 04033E ELSE IF SOURCE-WORD = ")"
    01 04033E PERFORM REGISTER-RIGHT-PAREN
    01 04033E PERFORM GET-WORD
  01 04033E ELSE MOVE 1 TO UNDE-SW
    01 04033E IF SOURCE-WORD-ITEM (1) = ""
      01 04033E MOVE SOURCE-WORD TO SAVE-WORD
      01 04033E ELSE MOVE 1 TO SUBSCRIPT-TEMP
      01 04033E PERFORM STEP-PAST-PARENS
      01 04033E UNTIL SOURCE-WORD-ITEM (SUBSCRIPT-TEMP) = ""
      01 04033E NOT = "("
      01 04033E UNSTRING SOURCE-WORD
      01 04033E DELIMITED BY ALL SPACES OR ""
    01 04033E INTO SAVE-WORD
    01 04033E DELIMITER IN SAVE-STOP
    01 04033E WITH POINTER SUBSCRIPT-TEMP
    01 04033E IF SAVE-STOP = ""
    01 04033E PERFORM REGISTER-RIGHT-PAREN.
  01 04030A STEP-PASTE-PARENS.
  01 04030A ADD 1 TO SUBSCRIPT-TEMP.
  01 04030A PERFORM REGISTER-LEFT-PAREN.
  01 04030A REGISTER-LEFT-PAREN.
  01 04030A REGISTER-RIGHT-PAREN.
  01 04030A REGISTER-LEFT-PAREN TO 07.
  01 04030A REGISTER-OPERATOR.
  01 04030A REGISTER-RIGHT-PAREN.
  01 04030A REGISTER-OPERATOR.
  01 04030A PERFORM REGISTER-OPERATOR.
  01 04030A REMARKED OUT SO THAT PARENTHESES ARE COUNTED AS PAIRS
/******************************************************
  * 4.2 REGISTER OPERATOR ROUTINE
  * ******************************************************
  * FUNCTIONS: ENTER THE OPERATOR CURRENTLY BEING SCANNED
  * INTO THE OPERATOR TABLE BY INCREMENTSING
  * THE COUNTER OF THE ENTRY ASSOCIATED WITH
  * THE PASSED INDEX
  * INPUT: TOTAL-FREQUENCY-OF-OPERATORS
  * OPERATOR-ERIES
  * OPERATOR-INDEX
  * OUTPUT: TOTAL-FREQUENCY-OF-OPERATORS
  * OPERATOR-ERIES
  * 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) = ZER0
ADD 1 TO NUMBER-OF-UNIQUE-OPERATORS.
ADD 1 TO OPERATOR-COUNTER (OPERATOR-INDEX).
FUNCTION: 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-

INPUT:
- NUMBER-OF-UNIQUE-OPERANDS
- TOTAL-FREQUENCY-OF-OPERANDS
- SAVE-WORD
- NUMBER-OF-LINKS

OUTPUT:
- OPERAND-FILE
- LINK-FILS
- NUMBER-OF-UNIQUE-OPERANDS
- NUMBER-OF-LINKS

LOCAL DATA:
- OPERAND-KEY
- OPERAND-FIND-FLAG
- OPERAND-COUNTER
- LAST-LINK
- LINK-KEY
- LINK-KEY
- CALLED-BY:
- COLLECT-OPERATORS-AND-OPERANDS
- CALLS: NONE

REGISTR-OPERAND,
ADD 1 TO TOTAL-FREQUENCY-OF-OPERANDS.
PERFORM DETERMINE-LINK-INDEX.

IF HEAD-LINK (LINK-INDEX) NOT EQUAL ZERO
MOVE HEAD-LINK (LINK-INDEX) TO LINK-KEY
MOVE ZERO TO OPERAND-FIND-FLAG
PERFORM GO-THRU-OPERANDS UNTIL LINK-KEY = ZERO
OK OPERAND-FOUND

IF OPERAND NOT FOUND
ADD 1 TO NUMBER-OF-LINKS
MOVE LAST-LINK TO LINK-KEY
MOVE NUMBER-OF-LINKS TO FORWARD-LINK
PERFORM REWRITE-LINK-FILE
MOVE NUMBER-OF-UNIQUE-OPERANDS TO LINK-KEY
MOVE ZERO TO FORWARD-LINK
ADD 1 TO NUMBER-OF-UNIQUE-OPERANDS
MOVE NUMBER-OF-UNIQUE-OPERANDS TO OPERAND-POINTER,
OPERAND-KEY
PERFORM WRITE-LINK-FILE
MOVE SAVE-WORD TO OPERAND-NAME
MOVE 1 TO OPERAND-COUNTER
PERFORM WRITE-OPERAND-FILE
1136#        ELSE ADD 1 TO OPERAND-COUNT
1137#            PERFORM WRITE-OPERAND-FILE
1138#        ELSE ADD 1 TO NUMER-D-OF-LINKS
1139#        MOVE NUMBER-OF-LINKS TO LINK-KEY
1140#            MOVE ZERO TO FORWARD-LINK
1141#            ADD 1 TO NUMBER-OF-UNIQUE-OPERANDS
1142#            MOVE NUMBER-OF-UNIQUE-OPERANDS TO OPERAND-POINTER
1143#            OPERAND-KEY
1144#            PERFORM WRITE-OPERAND-FILE
1145#            MOVE SAVD-WORD TO OPERAND-NAME
1146#            MOVE 1 TO UPERNAND-COUNT
1147#            PERFORM WRITE-OPERAND-FILE
1148#
1149#    DETERMINE-LINK-INDEX.
1150#    SET ALPHABET-INDEX TO 1
1151#    SEARCH ALPHABET-TABLE VARYING ALPHABET-INDEX
1152#    AT END MOVE 27 TO LINK-INDEX
1153#    WHEN ALPHABET-INDEX (ALPHABET-INDEX) = SAVE-WORD-ITEM (1)
1154#        MOVE ALPHABET-LINK (ALPHABET-INDEX) TO LINK-INDEX.
1155#
1156#    GO-THRU-OPERANDS.
1157#    MOVE LINK-KEY TO LAST-LINK.
1158#    READ LINK-FILE RECORD INTO OPERAND-ENTRY
1159#    INVALID KEY
1160#        DISPLAY "LINK-FILE SEARCH ERROR"
1161#        DISPLAY "LINK-KEY = ", LINK-KEY
1162#        DISPLAY "NUMBER OF LINKS = ", NUMBER-OF-LINKS
1163#        STOP RUN.
1164#    MOVE OPERAND-POINTER TO OPERAND-KEY.
1165#    READ OPERAND-FILE RECORD INTO OPERAND-ENTRY
1166#    INVALID KEY
1167#        DISPLAY "OPERAND-FILE SEARCH ERROR"
1168#        DISPLAY "OPERAND KEY = ", OPERAND-KEY
1169#        DISPLAY "NUMBER OF UNIQUE OPERANDS = ", NUMBER-OF-UNIQUE-OPERANDS
1170#        STOP RUN.
1171#    IF SAVE-ORD = OPERAND-NAME
1172#        DISPLAY "DUH! - ", LINE 15, POSITION 1
1173#        DISPLAY OPERAND-NAME, LINE 15, POSITION 9
1174#        DISPLAY "OPERAND KEY = ", LINE 16, POSITION 1
1175#        DISPLAY OPERAND-KEY, LINE 16, POSITION 15
1176#        MOVE 1 TO OPERAND-FIND-FLAG
1177#        ELSE MOVE FORWARD-LINK TO LINK-KEY.
1178#    WRITE-LINK-FILE.
1179#    WRITE OPERAND-LINK-RECORD
1180#    INVALID KEY
1181#        DISPLAY "RUN OUT OF SPACE IN LINK-FILE"
1182#        DISPLAY "LINK-KEY = ", LINK-KEY
1183#        STOP RUN.
1184#            REWRITE-LINK-FILE.
1185#            REWRITE OPERAND-LINK-RECORD
1186#    INVALID KEY
1187#        DISPLAY "PROBLEM REWRITING LINK-FILE"
13620  DISPLAY "LINK KEY =", LINK-KEY
13630  STOP RUN.
13640  * WRITE-OPERAND-FILE.
13650  WRITE OPERAND-RECORD FROM OPERAND-ENTRY
13660  INVALID KEY
13670  DISPLAY "RUN OUT OF SPACE IN OPERAND-FILE"
13680  DISPLAY "OPERAND KEY = ", OPERAND-KEY
13690  STOP RUN.
13700  * WRITE-OPERAND-FILE.
13710  REWRITE OPERAND-RECORD FROM OPERAND-ENTRY
13720  INVALID KEY
13730  DISPLAY "PROBLEM REWRITING OPERAND-FILE"
13740  DISPLAY "OPERAND KEY = ", OPERAND-KEY
13750  STOP RUN.
7 COMPUTE MEASURES ROUTINE

FUNCTION: COMPUTE THE METRICS DEFINED BY HALSTEAD AND HIS CO-WORKERS FROM THE INFORMATION COLLECTED.

INPUTS: NONE

OUTPUT: NONE

LOCAL DATA: NONE

CALLED BY: MAIN

CALLS:
- CALCULATE-PROGRAM-VOCABULARY
- CALCULATE-PROGRAM-LENGTH
- CALCULATE-PROGRAM-VOLUME
- CALCULATE-ESTIMATE-OF-LENGTH
- CALCULATE-IMPLEMENTATION-LEVEL
- CALCULATE-POTENTIAL-VOLUME
- CALCULATE-EFFORT-METRIC
- CALCULATE-PROGRAMMING-TIME
- CALCULATE-LANGUAGE-LEVEL

MEASURE-COMPUTATION SECTION 54.

PERFORM CALCULATE-PROGRAM-VOCABULARY.

PERFORM CALCULATE-PROGRAM-LENGTH.

PERFORM CALCULATE-PROGRAM-VOLUME.

PERFORM CALCULATE-ESTIMATE-OF-LENGTH.

PERFORM CALCULATE-IMPLEMENTATION-LEVEL.

PERFORM CALCULATE-POTENTIAL-VOLUME.

PERFORM CALCULATE-EFFORT-METRIC.

PERFORM CALCULATE-PROGRAMMING-TIME.

PERFORM CALCULATE-LANGUAGE-LEVEL.
**7.1 CALCULATE PROGRAM VOCABULARY ROUTINE**

**FUNCTION:** Compute the program vocabulary metric using the formula: \( N1 + N2 \)

**INPUT:**
- Number-of-unique-operators

**OUTPUT:**
- Number-of-unique-operators
- Program-vocabulary

**LOCAL DATA:** None

**called by:** Compute-measures

**calls:** None

---

`1414`  
`1415`  
`1416`  
`1417`  
`1418`  
`1419`  
`1420`  
`1421`  
`1422`  
`1423`  
`1424`  
`1425`  
`1426`  
`1427`  
`1428`  
`1429`  

`1430`  
`1431`  
`1432`  
`1433`
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
***************
7.3 CALCULATE ESTIMATE OF LENGTH ROUTINE

FUNCTION: COMPUTE THE ESTIMATE OF PROGRAM LENGTH
METRIC USING THE FORMULA:
N = IN1LOG2(IN2) + IN23LOG2(IN2)

INPUT: NUMBER-OF-UNIQUE-OPERATORS

OUTPUT: ESTIMATE-OF-PROGRAM-LENGTH

LOCAL DATA: PASSED-TO-LOG-Routine
TEMP-1
TEMP-7
TEMP-10

CALLED BY: COMPUTE-MEASURES

CALLS: COMPUTE-LOG-BASE-2WD

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

CALCULATE-ESTIMATE-OF-LENGTH.
MOVE NUMBER-OF-UNIQUE-OPERATORS TO PASSED-TO-LOG-Routine.
PERFORM COMPUTE-LOG-BASE-2WD.
MULTIPLY NUMBER-OF-UNIQUE-OPERATORS BY LOG-BASE2D-RESULT.
MOVE NUMBER-OF-UNIQUE-OPRANDS TO PASSED-TO-LOG-Routine.
PERFORM COMPUTE-LOG-BASE-2WD.
MULTIPLY NUMBER-OF-UNIQUE-OPRANDS BY LOG-BASE2D-RESULT.
GIVING TEMP-7.
ADD TEMP-1, TEMP-7.
GIVING ESTIMATE-OF-PROGRAM-LENGTH.
/*
 * CALCULATE PROGRAM VOLUME ROUTINE
 * FUNCTION: COMPUTE THE PROGRAM VOLUME METRIC USING THE
 * FORMULAS: V = (F1+F2)LOG2(N1+N2)
 * INPUT: PROGRAM-VOCABULARY
 * PROGRAM-LENGTH
 * OUTPUT: PROGRAM-VOLUME
 * CALLED BY: COMPUTE-MEASURES
 * CALLS: COMPUTE-LOG-TWO-RESULT
 */

1501 CALCULATE-PROGRAM-VOLUME.
1502 MOVE PROGRAM-VOCABULARY TO PASSED-TO-LOG ROUTINE.
1503 PERFORM COMPUTE-LOG-BASE-TWO.
1504 MULTIPY PROGRAM-LENGTH
1505 BY LOG-BASE-TWO-RESULT
1506 GIVING PROGRAM-VOLUME.
7.4.1 Compute Log Base Two Routine

Function: Calculate the base two logarithm of the

Input: Passed_to_log-routine

Output: Log_base_two_result

Local Data:

- LOG_ADDEND
- X_PLUS
- X_MINUS
- TEMP
- Multiplier
- K
- I
- LOG_E
- Fraction

Called By: Calculate_estimate_of_length

Calls: None

Compute-log-base-two

Mvul zero to half-log

Move 1 to log-addend

Add 1 to passed_to_log-routine

Giving x_plus

Subtract 1 from passed_to_log-routine

Giving x_minus

Divide x_minus by x_plus

Giving temp

Mvul temp to multiplier * half-log

Multiply temp by temp

Giving K

Perform compute_half-log

Varying i from 3 by 2

Until log_addend = 0

Multiply half-log by 2

Giving log-e

Multiply log-e by conversion-value

Giving log_base_two_result rounded

Compute-half-log

Divide 1 by i

Giving fraction rounded

Multiply multiplier by k

Giving multiplier

Multiply multiplier by fraction

Giving log_addend rounded

Add log_addend to half-log
7.5 Calculate implementation level routine

* Function: Compute the implementation level metric
  Using the formula
  \[ L = \frac{||z||_{L2}}{||z||_{L1}} \]

* Input: Number of unique operands
* Output: Implementation level

Local data: none
Called by: Compute measures
Called from: none

Calculate implementation level:

- Multiply number of unique operands by total frequency of operands
  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.
1000  FUNCTION CALCULATE POTENTIAL VOLUME ROUTINE
1001  FORMAT
1002  INPUT IMPLEMENTATION-LEVEL
1003  OUTPUT PROGRAM-VOLUME
1004  LOCAL DATA NONE
1005  CALLED SYS COMPUTE-MEASURES
1006  CALL SI NONE
1007  ** CALCULATE-POTENTIAL-VOLUME
1008  ** MULTIPLY IMPLEMENTATION-LEVEL
1009  ** BY PROGRAM-VOLUME
1010  ** GIVING POTENTIAL-VOLUME.
**FUNCTION:** COMPUTE THE EFFORT METRIC USING THE

**FORMULA:** EC = VSL

**INPUT:**
- TEMP-3
- PROGRAM-VOLUME

**OUTPUT:**
- EFFORT-METRIC
- TEMP-4

**LOCAL DATA:** TEMP-4

**CALLED BY:** COMPUTE-METRICES

**CALLS:** NONE

**CALCULATE EFFORT-METRIC:**

MULTIPLY TEMP-3 BY PROGRAM-VOLUME

GIVING TEMP-4.

DIVIDE TEMP-4 BY TEMP-2

GIVING EFFORT-METRIC ROUNDED.
/* 7.0 CALCULATE PROGRAMMING TIME ROUTINE */

FUNCTION: COMPUTE THE PROGRAMMING TIME METRIC IN TERMS OF SECONDS, MINUTES, AND HOURS USING STUDY'S NUMBER OF LB 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

*******************************************************************************/

1653*050132 CALCULATE-PROGRAMMING-TIME.
1655*050132 DIVIDE EFFORT-METRIC
1657*050132 BY 60
1659*050132 GIVING PROGRAMMING-TIME-SECONDS ROUNDED.
1661*050132 DIVIDE PROGRAMMING-TIME-SECONDS
1663*050132 BY 60
1665*050132 GIVING PROGRAMMING-TIME-HOURS ROUNDED.
/* 7.9 CALCULATE LANGUAGE LEVEL ROUTINE */

FUNCTION: COMPUTE THE LANGUAGE LEVEL MEASURES USING THE  
FORMULA (LIELEV)  

INPUT: IMPLEMENTATION-LEVEL  

OUTPUT: PROGRAM-VOLUME  

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.
**OUTPUT RESULTS ROUTINE**

FUNCTION: OUTPUT THE MEASURED AND COMPUTED METRICS TO THE PRINT-FILE

- INPUT: PRINT-LINE=1
- OUTPUT: PRINT-FILE
- LOCAL DATA: NONE
- CALLED BY: MAIN
- CALLS: OUTPUT-MEASURED-RESULTS
          OUTPUT-COMPUTED-RESULTS
          OUTPUT-OPERAND-FREQUENCIES
          OUTPUT-OPERATION-FREQUENCIES

**PRINT-RESULTS SECTION 59.**

- OUTPUT-RESULTS
  - WRITE PRINT-RECORD FROM PRINT-LINE=1
  - AFTER ADVANCING 4 LINES.
  - PERFORM OUTPUT-MEASURED-RESULT-1
  - THRU OUTPUT-MEASURED-RESULT-4
  - PERFORM OUTPUT-COMPUTED-RESULT-1
  - THRU OUTPUT-COMPUTED-RESULT-9
  - PERFORM OUTPUT-OPERATOR-FREQUENCIES
  - PERFORM OUTPUT-OPERAND-FREQUENCIES
/* 8.1 OUTPUT MEASURED RESULTS ROUTINE */

FUNCTION: OUTPUT THE RESULTS OF THE MEASURED METRICS

INPUT:
- NUMBER-OF-UNIQUE-OPERATORS
- PRINT-LINE-1
- NUMBER-OF-UNIQUE-OPERANDS
- PRINT-LINE-2

OUTPUT:
- PRINT-LINE-3
- TOTAL-FREQUENCY-OF-OPERATORS
- PRINT-LINE-4
- TOTAL-FREQUENCY-OF-OPERANDS
- PRINT-LINE-5

LOCAL DATA: NONE

CALLED BY: OUTPUT-RESULTS

CALLS: NONE

OUTPUT-MEASURED-RESULT-1:

OUTPUT-MEASURED-RESULT-2:

OUTPUT-MEASURED-RESULT-3:

OUTPUT-MEASURED-RESULT-4:
FUNCTIONS OUTPUT THE RESULTS OF THE COMPUTED METRICS TO THE PRINT-FILE

INPUTS PROGRAM-VOCABULARY
       PRINT-LINE-6
       PRINT-LINE-7
       PROGRAM-LENGTH
       PROGRAM-VOLUME
       IMPLEMENTATION-LEVEL
       PRINT-LINE-10
       POTENTIAL-VOLUME
       PRINT-LINE-11
       EFFICIENCY-INDEX
       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

OUTPUTS PRINT-FILE
       LOCAL DATA NONE
       CALLED BY OUTPUT-RESULTS
       CALLED 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.
OUTPUT-COMPUTED-RESULT-5.

MOVE IMPLEMENTATION-LEVEL TO OUTPUT-VALUE-10.
WRITE PRINT-RECORD FROM PRINT-LINE-10
AFTER ADVANCING 2 LINES.

OUTPUT-COMPUTED-RESULT-6.
MOVE POTENTIAL-VOLUME TO OUTPUT-VALUE-11.
WRITE PRINT-RECORD FROM PRINT-LINE-11
AFTER ADVANCING 2 LINES.

OUTPUT-COMPUTED-RESULT-7.
MOVE EFFORT-METRIC TO OUTPUT-VALUE-12.
WRITE PRINT-RECORD FROM PRINT-LINE-12
AFTER ADVANCING 2 LINES.

OUTPUT-COMPUTED-RESULT-8.
MOVE PROGRAMMING-TIME-SECONDS TO OUTPUT-VALUE-13.
WRITE PRINT-RECORD FROM PRINT-LINE-13
AFTER ADVANCING 2 LINES.

OUTPUT-COMPUTED-RESULT-9.
MOVE LANGUAGE-LEVEL TO OUTPUT-VALUE-16.
WRITE PRINT-RECORD FROM PRINT-LINE-16
AFTER ADVANCING 2 LINES.
FUNCTION: OUTPUT THE OPERATORS FOUND IN THE SOURCE PROGRAM IN ORDER OF DESCENDING FREQUENCY TO THE PRINT-FILE.

INPUTS: PAINT-LINE-17.

OUTPUTS: PRINT-FILE.

LOCAL DATA: NUMBER-OF-ITEMS-PRINTED.

LAST-ONE-PRINTED.

LAST-HIGH.

ITEM-ENDING-CONSIDERED.

ONE-TO-PRINT.

CURRENT-HIGH.

UPGRAPHR-OUTPUT-COUNTER.

OUTPUT-RESULTS.

CALLS: GET-TO-TOP-OF-PAGE.

PRINT-BLANK-LINE.

OUTPUT-OPERATOR-FREQUENCIES.

PERFORM GET-TO-TOP-OF-PAGE.

WRITE PRINT-RECORD FROM PRINT-LINE-17.

AFTER ADVANCING 3 LINES.

MOVE 9999999999999999999999 TO LAST-HIGH.

MOVE 9999999999999999999999 TO LAST-HIGH.

MOVE 9999999999999999999999 TO LAST-HIGH.

PERFORM OUTPUT-AN-OPERATOR.

UNTIL NUMBER-OF-ITEMS-PRINTED = NUMBER-OF-UNIQUE-OPERATORS.

OUTPUT-AN-OPERATOR.

MOVE 9999999999999999999999 TO ITEM-ENDING-CONSIDERED.

CURRENT-HIGH.

DONT-SW.

PERFORM LOOK-FOR-HIGH-OPERATOR-COUNTER.

UNTIL ITEM-ENDING-CONSIDERED = 98 OR DONT.

MOVE ONE-TO-PRINT TO LAST-ONE-PRINTED.

MOVE CURRENT-HIGH TO LAST-HIGH.

UPGRAPHR-OUTPUT-COUNTER.

MOVE OPERATOR-ELEMENT (ONE-TO-PRINT) TO OPERATOR-OUTPUT-ELEMENT.

WRITE PRINT-RECORD FROM PRINT-LINE-17.

AFTER ADVANCING 2 LINES.

AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE.

THRU PRINT-BLANK-LINE.

ADD 1 TO NUMBER-OF-ITEMS-PRINTED.

LOOK-FOR-HIGH-OPERATOR-COUNTER.
ADD 1 TO ITEM-BEING-CONSIDERED.

IF OPERATOR-COUNTERITEM-BEING-CONSIDERED) > CURRENT-HIGH
    AND OPERATOR-COUNTERITEM-BEING-CONSIDERED) < LAST-HIGH
    MOVE ITEM-BEING-CONSIDERED TO ONE-TO-PRINT
    MOVE OPERATOR-COUNTERITEM-BEING-CONSIDERED) TO CURRENT-HIGH
    ELSE IF OPERATOR-COUNTERITEM-BEING-CONSIDERED) = LAST-HIGH
    AND ITEM-BEING-CONSIDERED > LAST-ONE-PRINTED
    MOVE ITEM-BEING-CONSIDERED TO ONE-TO-PRINT
    MOVE OPERATOR-COUNTERITEM-BEING-CONSIDERED) TO CURRENT-HIGH
    MOVE 1 TO DONE-SW.
/* 8.4 OUTPUT-OPERAND-FREQUENCIES */

FUNCTION: OUTPUT THE OPERANDS IN THE SOURCE PROGRAM IN ORDER OF DESCENDING FREQUENCY TO THE PRINT-FILE.

INPUT: PRINT-LINE-19

LOCAL DATA: NUMBER-OF-UNIQUE-OPERANDS

OUTPUT: PRINT-FILE-20

OUTPUT-OPERAND-KEY

CALLED BY: OUTPUT-RESULTS

OUTPUT-OPERAND-FREQUENCIES.
PERFORM GET-TO-TOP-OF-PAGE.
WRITE PRINT-RECORD FROM PRINT-LINE-19 AFTER ADVANCING 3 LINES.
MOVE ZER0 TO NUMBER-OF-ITEMS-PRINTED,
LAST-ONE-PRINTED.
MOVE YYYYYYYYYYYYYYY YYYYYYYYYYYYYY TO LAST-HIGH.
PERFORM OUTPUT-AN-OPERAND UNTIL NUMBER-OF-ITEMS-PRINTED = NUMBER-OF-UNIQUE-OPERANDS.

OUTPUT-AN-OPERAND.
MOVE ZER0 TO OPERAND-KEY,
CURRENT-HIGH.
DUNE-SW.
PERFORM LOOK-FOR-HIGH-OPERAND-COUNTER UNTIL OPERAND-KEY = NUMBER-OF-UNIQUE-OPERANDS OR DUNE.
MOVE ONE-TO-PRINT TO LAST-ONE-PRINTED,
OPERAND-KEY.
MOVE CURRENT-HIGH TO LAST-HIGH.
OPERAND-OUTPUT-COUNTER.
READ OPERAND-FILE RECORD INTO OPERAND-ENTRY
INVALID KEY.
DISPLAY "OPERAND-FILE SEARCH ERROR"
STOP RUN.
MOVE OPERAND-NAME TO OPERAND-OUTPUT-COUNTER.
WRITE PRINT-RECORD FROM PRINT-LINE-19 AFTER ADVANCING 2 LINES
AT END-OF-PAGE PERFORM GET-TO-TOP-OF-PAGE THRU PRINT-BLANK-LINE.
ADD 1 TO NUMBER-OF-ITEMS-PRINTED.

LOOK-FOR-HIGH-OPR-AND-COUNTER.

ADD 1 TO OPERAND-KEY.

READ OPERAND-FILE RECORD INTO OPERAND-ENTRY

INVALID KEY

DISPLAY "OPERAND-FILE SEARCH ERROR"

STOP RUN.

IF OPERAND-COUNTER > CURRENT-HIGH

AND OPERAND-COUNTER < LAST-HIGH

MOVE OPERAND-KEY TO ONE-TO-PRINT

MOVE OPERAND-COUNTER TO CURRENT-HIGH

ELSE IF OPERAND-COUNTER = LAST-HIGH

AND OPERAND-KEY > LAST-ONE-PRINTED

MOVE OPERAND-KEY TO ONE-TO-PRINT

MOVE OPERAND-COUNTER TO CURRENT-HIGH

MOVE 1 TO DONE-SW.

ZZZZZ END PROGRAM. ****END OF FILE****
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>SIZE</th>
<th>DEBUG</th>
<th>TYPE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>480</td>
<td>ANS</td>
<td>0</td>
<td>FILE</td>
</tr>
<tr>
<td>0040</td>
<td>512</td>
<td>ANS</td>
<td>0</td>
<td>FILE</td>
</tr>
<tr>
<td>0080</td>
<td>132</td>
<td>ANS</td>
<td>0</td>
<td>FILE</td>
</tr>
<tr>
<td>0120</td>
<td>504</td>
<td>GRP</td>
<td>0</td>
<td>FILE</td>
</tr>
<tr>
<td>0160</td>
<td>140</td>
<td>GRP</td>
<td>0</td>
<td>FILE</td>
</tr>
<tr>
<td>0200</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>FILE</td>
</tr>
<tr>
<td>0240</td>
<td>6</td>
<td>ANS</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>0280</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>0300</td>
<td>1</td>
<td>ANS</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>0340</td>
<td>65</td>
<td>ANS</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>0380</td>
<td>8</td>
<td>ANS</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>0420</td>
<td>40</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>0460</td>
<td>1</td>
<td>GRP</td>
<td>1</td>
<td>GROUP</td>
</tr>
<tr>
<td>0500</td>
<td>122</td>
<td>ANS</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>0540</td>
<td>1</td>
<td>ANS</td>
<td>1</td>
<td>GROUP</td>
</tr>
<tr>
<td>0580</td>
<td>122</td>
<td>ANS</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>0620</td>
<td>1</td>
<td>ANS</td>
<td>1</td>
<td>GROUP</td>
</tr>
<tr>
<td>0660</td>
<td>470</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>06A0</td>
<td>4</td>
<td>GRP</td>
<td>1</td>
<td>GROUP</td>
</tr>
<tr>
<td>0740</td>
<td>3</td>
<td>ANS</td>
<td>1</td>
<td>GROUP</td>
</tr>
<tr>
<td>0780</td>
<td>17</td>
<td>NSU</td>
<td>1</td>
<td>GROUP</td>
</tr>
<tr>
<td>1120</td>
<td>72</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
</tr>
<tr>
<td>1160</td>
<td>3</td>
<td>GRP</td>
<td>1</td>
<td>GROUP</td>
</tr>
<tr>
<td>1200</td>
<td>1</td>
<td>ANS</td>
<td>1</td>
<td>GROUP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR</th>
<th>SIZE</th>
<th>DEBUG</th>
<th>UNDER</th>
<th>TYPE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>167F</td>
<td>2</td>
<td>NSE</td>
<td>1</td>
<td>NUMERIC</td>
<td>ALPHABET-LINK</td>
</tr>
<tr>
<td>168C</td>
<td>4</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>OPERAND-HEAD-LINKS</td>
</tr>
<tr>
<td>168C</td>
<td>1</td>
<td>GRP</td>
<td>1</td>
<td>GROUP</td>
<td>HEAD-LINK-LIST</td>
</tr>
<tr>
<td>168C</td>
<td>1</td>
<td>NSE</td>
<td>1</td>
<td>NUMERIC</td>
<td>HEAD-LINK</td>
</tr>
<tr>
<td>180F</td>
<td>14</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>OPERAND-ENTRY</td>
</tr>
<tr>
<td>1822</td>
<td>12</td>
<td>ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>OPERAND-NAME</td>
</tr>
<tr>
<td>1B9C</td>
<td>13</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OPERAND-COUNTER</td>
</tr>
<tr>
<td>195E</td>
<td>6</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>THE-TIME</td>
</tr>
<tr>
<td>195E</td>
<td>2</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>HOURS</td>
</tr>
<tr>
<td>195E</td>
<td>2</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>MINUTES</td>
</tr>
<tr>
<td>1962</td>
<td>2</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>SECONDS</td>
</tr>
<tr>
<td>1964</td>
<td>123</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>HEADER</td>
</tr>
<tr>
<td>197A</td>
<td>30</td>
<td>ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>PROGRAM-NAME</td>
</tr>
<tr>
<td>1B8D</td>
<td>9</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>DATE-OUT</td>
</tr>
<tr>
<td>1BCA</td>
<td>2</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>HOURS-OUT</td>
</tr>
<tr>
<td>1BCD</td>
<td>2</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>MINUTES-OUT</td>
</tr>
<tr>
<td>1B9D</td>
<td>2</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>SECONDS-OUT</td>
</tr>
<tr>
<td>19DC</td>
<td>3</td>
<td>ANS</td>
<td>0</td>
<td>ALPHANUMERIC</td>
<td>PAGE-NUMBRL</td>
</tr>
<tr>
<td>180E</td>
<td>119</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-1</td>
</tr>
<tr>
<td>1BD9</td>
<td>116</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-2</td>
</tr>
<tr>
<td>1BAD</td>
<td>16</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-2</td>
</tr>
<tr>
<td>1ACC</td>
<td>116</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-3</td>
</tr>
<tr>
<td>1B3E</td>
<td>16</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-3</td>
</tr>
<tr>
<td>1B4D</td>
<td>116</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-4</td>
</tr>
<tr>
<td>1B42</td>
<td>16</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-4</td>
</tr>
<tr>
<td>1B44</td>
<td>116</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-5</td>
</tr>
<tr>
<td>1C16</td>
<td>16</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-5</td>
</tr>
<tr>
<td>1C1B</td>
<td>116</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-6</td>
</tr>
<tr>
<td>1C8A</td>
<td>16</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-6</td>
</tr>
<tr>
<td>1C9C</td>
<td>116</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-7</td>
</tr>
<tr>
<td>1CFE</td>
<td>16</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-7</td>
</tr>
<tr>
<td>1D10</td>
<td>117</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-8</td>
</tr>
<tr>
<td>1DFB</td>
<td>12</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-8</td>
</tr>
<tr>
<td>1D98</td>
<td>121</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-9</td>
</tr>
<tr>
<td>1DF9</td>
<td>14</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-9</td>
</tr>
<tr>
<td>1E02</td>
<td>121</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-10</td>
</tr>
<tr>
<td>1E6D</td>
<td>14</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-10</td>
</tr>
<tr>
<td>1E7C</td>
<td>121</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-11</td>
</tr>
<tr>
<td>1E67</td>
<td>14</td>
<td>NSE</td>
<td>0</td>
<td>NUMERIC</td>
<td>OUTPUT-VALUE-11</td>
</tr>
<tr>
<td>1EF6</td>
<td>121</td>
<td>GRP</td>
<td>0</td>
<td>GROUP</td>
<td>PRINT-LINE-12</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>SIZE</td>
<td>TYPE</td>
<td>NAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------</td>
<td>-------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1051</td>
<td>14 NS</td>
<td>GROUP</td>
<td>OUTPUT-VALUE-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1170</td>
<td>14 NS</td>
<td>GROUP</td>
<td>PRINT-LINE-13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1199</td>
<td>14 NS</td>
<td>GROUP</td>
<td>PRINT-LINE-14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2039</td>
<td>14 NS</td>
<td>GROUP</td>
<td>PRINT-LINE-15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2066</td>
<td>14 NS</td>
<td>GROUP</td>
<td>PRINT-LINE-16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2084</td>
<td>14 NS</td>
<td>GROUP</td>
<td>PRINT-LINE-17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2119</td>
<td>12 AN</td>
<td>ALPHANUMERIC</td>
<td>OPERATOR-OUTPUT-ELEMENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2264</td>
<td>6 NS</td>
<td>ALPHANUMERIC</td>
<td>THE-DATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2360</td>
<td>4 NS</td>
<td>ALPHANUMERIC</td>
<td>LINE-NUMBER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td>SIZE</td>
<td>DEBUG</td>
<td>ORDER</td>
<td>TYPE</td>
<td>NAME</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>2371</td>
<td>4</td>
<td>NSP</td>
<td>0</td>
<td>NUMERIC EDITED</td>
<td>LINE-NUMBER-DISPLAY</td>
</tr>
<tr>
<td>2375</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>OPERAND-KEY</td>
</tr>
<tr>
<td>2387</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LINK-KEY</td>
</tr>
<tr>
<td>2399</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>NUMBER-OF-LINKS</td>
</tr>
<tr>
<td>2408</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LAST-LINK</td>
</tr>
<tr>
<td>2390</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LINK-INDEX</td>
</tr>
<tr>
<td>23CF</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>ITEM-BEING-CONSIDERED</td>
</tr>
<tr>
<td>23E1</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>ONE-TO-PRINT</td>
</tr>
<tr>
<td>23F3</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LAST-ONE-PRINTED</td>
</tr>
<tr>
<td>2405</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>CURRENT-HIGH</td>
</tr>
<tr>
<td>2417</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LAST-HIGH</td>
</tr>
<tr>
<td>2429</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>NUMBER-OF-ITEMS-PRINTED</td>
</tr>
<tr>
<td>2438</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>NUMBER-OF-UNIQUE-OPERANDS</td>
</tr>
<tr>
<td>2440</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>NUMBER-OF-UNIQUE-OPERATORS</td>
</tr>
<tr>
<td>245F</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TOTAL-FREQUENCY-OF-OPERANDS</td>
</tr>
<tr>
<td>2471</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TOTAL-FREQUENCY-OF-OPERATORS</td>
</tr>
<tr>
<td>2443</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PROGRAM-VOCABULARY</td>
</tr>
<tr>
<td>2495</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PROGRAM-LENGTH</td>
</tr>
<tr>
<td>2447</td>
<td>16</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PROGRAM-VOLUME</td>
</tr>
<tr>
<td>2487</td>
<td>6</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>IMPLEMENTATION-LEVEL</td>
</tr>
<tr>
<td>2480</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>POTENTIAL-VOLUME</td>
</tr>
<tr>
<td>24CF</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>EFFORT-METRIC</td>
</tr>
<tr>
<td>24E1</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>ESTIMATE-OF-PROGRAM-LENGTH</td>
</tr>
<tr>
<td>24F3</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PROGRAMMING-TIME-SECONDS</td>
</tr>
<tr>
<td>2505</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PROGRAMMING-TIME-MINUTES</td>
</tr>
<tr>
<td>2517</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PROGRAMMING-TIME-HOURS</td>
</tr>
<tr>
<td>2529</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LANGUAGE-LEVEL</td>
</tr>
<tr>
<td>2530</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>PASSED-TO-LOG-Routine</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>SIZE</td>
<td>DEBUG</td>
<td>ORDER</td>
<td>TYPE</td>
<td>NAME</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------</td>
<td>--------</td>
<td>------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>2540</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LOG-RESULT</td>
</tr>
<tr>
<td>255F</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TEMP-1</td>
</tr>
<tr>
<td>2571</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TEMP-2</td>
</tr>
<tr>
<td>2583</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TEMP-3</td>
</tr>
<tr>
<td>2595</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TEMP-4</td>
</tr>
<tr>
<td>25A7</td>
<td>10</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TEMP-5</td>
</tr>
<tr>
<td>25B9</td>
<td>14</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TEMP-6</td>
</tr>
<tr>
<td>25C7</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>TEMP-7</td>
</tr>
<tr>
<td>25D9</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>HALF-LOG</td>
</tr>
<tr>
<td>25E0</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LOG-ADDEND</td>
</tr>
<tr>
<td>25F0</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>LOG-E</td>
</tr>
<tr>
<td>260F</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>MULTIPLIER</td>
</tr>
<tr>
<td>2621</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>FRACTION</td>
</tr>
<tr>
<td>2633</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>I</td>
</tr>
<tr>
<td>2645</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>K</td>
</tr>
<tr>
<td>2657</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>XPLUS-1</td>
</tr>
<tr>
<td>2669</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>X-MINUS-1</td>
</tr>
<tr>
<td>2678</td>
<td>18</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>CONVERSION-VALUE</td>
</tr>
<tr>
<td>2680</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>FILE-FLAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONDITION-NAME</td>
<td>FIXED-LENGTH-FILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONDITION-NAME</td>
<td>VARIABLE-LENGTH-FILE</td>
</tr>
<tr>
<td>268E</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>EDF-FLAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONDITION-NAME</td>
<td>NOT-END-OF-FILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONDITION-NAME</td>
<td>END-OF-FILE</td>
</tr>
<tr>
<td>269F</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>AREA-A-FLAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONDITION-NAME</td>
<td>AREA-A-OCUPIED</td>
</tr>
<tr>
<td>2690</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>CONSUME-SW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONDITION-NAME</td>
<td>CONSUME-DONE</td>
</tr>
<tr>
<td>2691</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>QUOTE-SW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONDITION-NAME</td>
<td>BALANCED-QUOTE</td>
</tr>
<tr>
<td>2692</td>
<td>1</td>
<td>NSU</td>
<td>0</td>
<td>NUMERIC</td>
<td>DONE-SW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONDITION-NAME</td>
<td>DONE</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Number of Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>84</td>
</tr>
<tr>
<td>2</td>
<td>67</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure G.1 Table of Cyclomatic Complexity Values Found in the First Version of Program A.
<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Number of Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>111</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure G.2 Table of Cyclomatic Complexity Values Found in the Second Version of Program A.
<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Number of Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73</td>
</tr>
<tr>
<td>2</td>
<td>59</td>
</tr>
<tr>
<td>3</td>
<td>44</td>
</tr>
<tr>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure G.3  Table of Cyclomatic Complexity Values Found in the First Version of Program B.
<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Number of Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>75</td>
</tr>
<tr>
<td>3</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure G.4 Table of Cyclomatic Complexity Values Found in the Second Version of Program B.
<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Number of Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>79</td>
</tr>
<tr>
<td>2</td>
<td>67</td>
</tr>
<tr>
<td>3</td>
<td>82</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure G.5 Table of Cyclomatic Complexity Values Found in the First Version of Program C.
<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Number of Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>79</td>
</tr>
<tr>
<td>2</td>
<td>66</td>
</tr>
<tr>
<td>3</td>
<td>82</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured Properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Unique Operators</td>
<td>M1</td>
<td>60</td>
</tr>
<tr>
<td>Number of Unique Operands</td>
<td>M2</td>
<td>1011</td>
</tr>
<tr>
<td>Total Frequency of Operators</td>
<td>F1</td>
<td>5702</td>
</tr>
<tr>
<td>Total Frequency of Operands</td>
<td>F2</td>
<td>6220</td>
</tr>
<tr>
<td>Computed Measures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program Vocabulary</td>
<td>M1+M2</td>
<td>1073</td>
</tr>
<tr>
<td>Program length</td>
<td>F1+F2</td>
<td>11922</td>
</tr>
<tr>
<td>Estimated Program Length</td>
<td>N* = (M1)log2(M1)+(M2)log2(M2)</td>
<td>10468.62</td>
</tr>
<tr>
<td>Program Volume</td>
<td>V = [F1+F2]log2(M1+M2)</td>
<td>120023.9524</td>
</tr>
<tr>
<td>Implementation Level</td>
<td>L = APPR. [2(N2)/(F1+F2)]</td>
<td>.0054</td>
</tr>
<tr>
<td>Potential Volume</td>
<td>V* = [L][V]</td>
<td>648.1293</td>
</tr>
<tr>
<td>Effort Metric</td>
<td>E = V/L</td>
<td></td>
</tr>
<tr>
<td>Programming Time</td>
<td>T = E/[A]</td>
<td>2210909.5435</td>
</tr>
<tr>
<td>Language Level</td>
<td>[L][L][V]</td>
<td>3.4998</td>
</tr>
</tbody>
</table>
FREQUENCY OF OPERATORS IN DESCENDING ORDER:

1131 TO
958 MOVE
629 =
606 PERFORM
403 IF
238 ELSE
172 NOT
149 DISPLAY
97 OR
92 UNTIL
90 OF
82 END
82 (}
72 ADD
54 SET
53 >
48 KEY
46 ACCEPT
44 BY
41 GIVING
38 READ
36 SUBTRACT
35 INTO
33 NEXT
31 INVALID
29 AND
29 READ
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.
<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured Properties:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Unique Operators</td>
<td>N1</td>
<td>61</td>
</tr>
<tr>
<td>Number of Unique Operands</td>
<td>N2</td>
<td>1204</td>
</tr>
<tr>
<td>Total Frequency of Operators</td>
<td>F1</td>
<td>7355</td>
</tr>
<tr>
<td>Total Frequency of Operands</td>
<td>F2</td>
<td>8026</td>
</tr>
<tr>
<td>Computed Measures:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program Vocabulary</td>
<td>N1+N2</td>
<td>1265</td>
</tr>
<tr>
<td>Program Length</td>
<td>F1+F2</td>
<td>15361</td>
</tr>
<tr>
<td>Estimated Program Length</td>
<td>N* = [IN1LOG2(N1)+IN2LOG2(N2)]</td>
<td>12683.05</td>
</tr>
<tr>
<td>Program Volume</td>
<td>V = [(F1+F2)LOG2(N1+N2)]</td>
<td>158500.0001</td>
</tr>
<tr>
<td>Implementation Level</td>
<td>L = APPR. (2[N2]+IN1)[IN1][F2]</td>
<td>.0049</td>
</tr>
<tr>
<td>Potential Volume</td>
<td>V* = [LJV1]</td>
<td>7764500</td>
</tr>
<tr>
<td>Effort Metric</td>
<td>EC = V/L</td>
<td>32225656.5814</td>
</tr>
<tr>
<td>Programming Time</td>
<td>T = EC/18</td>
<td>1790314.2545 sec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29038.570 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4973095 max.</td>
</tr>
<tr>
<td>Language Level</td>
<td>[L][L][V]</td>
<td>3.6055</td>
</tr>
</tbody>
</table>
FREQUENCY OF OPERATORS IN DESCENDING ORDER:

1435  TO
1221  MOVE
  981  *
  779  PERFORM
  639  IF
  390  ELSE
  210  NOT
  171  DISPLAY
  134  I
  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    C
19    AFTER
16    ADVANCING
17    DELIMITED
16    VARYING
14    UP
12    STRING
10    WITH
7    MULTIPLY
6    DIVIDE
5    STOP
4    INSPECT
4    OUTPUT
4    ECHO
4    EXDR
3    ALL
3    SEARCH
3    TALLYING
2    DELIMITER
2    IN
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.
<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MEASURED PROPERTIES:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMBER OF UNIQUE OPERATORS</td>
<td>M1</td>
<td>47</td>
</tr>
<tr>
<td>NUMBER OF UNIQUE OPERANDS</td>
<td>N2</td>
<td>744</td>
</tr>
<tr>
<td>TOTAL FREQUENCY OF OPERATORS</td>
<td>F1</td>
<td>3595</td>
</tr>
<tr>
<td>TOTAL FREQUENCY OF OPERANDS</td>
<td>F2</td>
<td>3760</td>
</tr>
<tr>
<td><strong>COMPUTED MEASURES:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROGRAM VOCABULARY</td>
<td>M1+N2</td>
<td>791</td>
</tr>
<tr>
<td>PROGRAM LENGTH</td>
<td>F1+F2</td>
<td>7355</td>
</tr>
<tr>
<td>ESTIMATED PROGRAM LENGTH</td>
<td>N* (= {N1}log_2{N1}+{N2}log_2{N2} )</td>
<td>7798.19</td>
</tr>
<tr>
<td>PROGRAM VOLUME</td>
<td>V (= {F1+F2}log_2{N1+N2} )</td>
<td>70810.5117</td>
</tr>
<tr>
<td>IMPLEMENTATION LEVEL</td>
<td>L (= \text{APPR}., {N1N2}/{N1+N2}{F1} )</td>
<td>0.0089</td>
</tr>
<tr>
<td>POTENTIAL VOLUME</td>
<td>V* (= {L}I{V} )</td>
<td>594.8082</td>
</tr>
<tr>
<td>EFFORT METRIC</td>
<td>EC (= V/L )</td>
<td>8469700.0181</td>
</tr>
<tr>
<td>PROGRAMMING TIME</td>
<td>T (= EC/30 )</td>
<td>467205.5556 SEC.</td>
</tr>
<tr>
<td>LANGUAGE LEVEL</td>
<td>[L][L][L][V]</td>
<td>4.9963</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7786.7593 MIN.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1219.7793 HRS.</td>
</tr>
</tbody>
</table>
HALSTEAD'S COMPLEXITY MEASURE

PROGRAM NAME = PROGRAM-B-VERSION-1

FREQUENCY OF OPERATORS IN DESCENDING ORDER:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO</td>
<td>580</td>
</tr>
<tr>
<td>MOVE</td>
<td>567</td>
</tr>
<tr>
<td>PERFORM</td>
<td>439</td>
</tr>
<tr>
<td>-</td>
<td>418</td>
</tr>
<tr>
<td>IF</td>
<td>313</td>
</tr>
<tr>
<td>ELSE</td>
<td>191</td>
</tr>
<tr>
<td>NOT</td>
<td>89</td>
</tr>
<tr>
<td>=</td>
<td>79</td>
</tr>
<tr>
<td>UNTIL</td>
<td>77</td>
</tr>
<tr>
<td>OR</td>
<td>70</td>
</tr>
<tr>
<td>GD</td>
<td>62</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>53</td>
</tr>
<tr>
<td>BY</td>
<td>52</td>
</tr>
<tr>
<td>&gt;</td>
<td>51</td>
</tr>
<tr>
<td>ADD</td>
<td>44</td>
</tr>
<tr>
<td>SUBTRACT</td>
<td>43</td>
</tr>
<tr>
<td>VARYING</td>
<td>32</td>
</tr>
<tr>
<td>GIVING</td>
<td>30</td>
</tr>
<tr>
<td>AND</td>
<td>24</td>
</tr>
<tr>
<td>SET</td>
<td>24</td>
</tr>
<tr>
<td>INTO</td>
<td>23</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>19</td>
</tr>
<tr>
<td>PROMPT</td>
<td>17</td>
</tr>
<tr>
<td>INVALID</td>
<td>15</td>
</tr>
<tr>
<td>OPEN</td>
<td>15</td>
</tr>
<tr>
<td>ERASE</td>
<td>15</td>
</tr>
<tr>
<td>READ</td>
<td>13</td>
</tr>
<tr>
<td>Token</td>
<td>Count</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>UNSTRING</td>
<td>11</td>
</tr>
<tr>
<td>UP</td>
<td>13</td>
</tr>
<tr>
<td>CLOSE</td>
<td>11</td>
</tr>
<tr>
<td>KEY</td>
<td>8</td>
</tr>
<tr>
<td>DELIMITED</td>
<td>7</td>
</tr>
<tr>
<td>NEXT</td>
<td>7</td>
</tr>
<tr>
<td>STRING</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>5</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>4</td>
</tr>
<tr>
<td>REWRITE</td>
<td>3</td>
</tr>
<tr>
<td>WITH</td>
<td>2</td>
</tr>
<tr>
<td>AFTER</td>
<td>2</td>
</tr>
<tr>
<td>MULTIPLY</td>
<td>2</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>2</td>
</tr>
<tr>
<td>GREATER</td>
<td>1</td>
</tr>
<tr>
<td>STOP</td>
<td>1</td>
</tr>
<tr>
<td>THAN</td>
<td>1</td>
</tr>
<tr>
<td>EXON</td>
<td>1</td>
</tr>
<tr>
<td>SHIFT</td>
<td>1</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER OF UNIQUE OPERATORS</td>
<td>N1</td>
<td>48</td>
</tr>
<tr>
<td>NUMBER OF UNIQUE OPERANDS</td>
<td>N2</td>
<td>845</td>
</tr>
<tr>
<td>TOTAL FREQUENCY OF OPERATORS</td>
<td>F1</td>
<td>4430</td>
</tr>
<tr>
<td>TOTAL FREQUENCY OF OPERANDS</td>
<td>F2</td>
<td>4470</td>
</tr>
<tr>
<td>PROGRAM VOCABULARY</td>
<td>N1+N2</td>
<td>693</td>
</tr>
<tr>
<td>PROGRAM LENGTH</td>
<td>F1+F2</td>
<td>9108</td>
</tr>
<tr>
<td>ESTIMATED PROGRAM LENGTH</td>
<td>N0 = ( \lceil N1 \log_2(N1) + N2 \log_2(N2) \rceil )</td>
<td>8483.85</td>
</tr>
<tr>
<td>PROGRAM VOLUME</td>
<td>V = ( \lceil F1 + F2 \log_2(N1+N2) \rceil )</td>
<td>89281.3190</td>
</tr>
<tr>
<td>IMPLEMENTATION LEVEL</td>
<td>L = ( \text{APPR.} { 2(N2) + { \text{N1} } (\text{F2}) } )</td>
<td>.0075</td>
</tr>
<tr>
<td>POTENTIAL VOLUME</td>
<td>V0 = ( \lceil L \rceil (V) )</td>
<td>669.4098</td>
</tr>
<tr>
<td>EFFORT METRIC</td>
<td>EC = V/L</td>
<td>-11862476.0314</td>
</tr>
<tr>
<td>PROGRAMMING TIME</td>
<td>T = EC/10</td>
<td>659026.4462 SEC.</td>
</tr>
<tr>
<td>LANGUAGE LEVEL</td>
<td>( \lceil L \rceil (L) (V) )</td>
<td>5.0220</td>
</tr>
<tr>
<td>Frequency of Operators in Descending Order</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>870</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>862</td>
<td>MOVE</td>
<td></td>
</tr>
<tr>
<td>514</td>
<td>PERFORM</td>
<td></td>
</tr>
<tr>
<td>492</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>303</td>
<td>IF</td>
<td></td>
</tr>
<tr>
<td>229</td>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>GO</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>NOT</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>UNTIL</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>SUBTRACT</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>DISPLAY</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>GIVING</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>OPEN</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>VARYING</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>INTO</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>ACCEPT</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>CLOSE</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>PROMPT</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>INVALID</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>NEXT</td>
<td></td>
</tr>
</tbody>
</table>

- K-2 -
17. READ
15. DIVIDE
15. UNSTRING
14. ERASE
13. UP
12. KEY
11. TIMES
10. DELIMITED
9. REWRITE
9. STRING
9. <
8. MULTIPLY
7. COMPUTE
6. AFTER
6. WITH
5. OUTPUT
5. STOP
4. GREATER
4. THAN
4. XOR
4. 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.
<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured Properties:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Unique Operators</td>
<td>( N_1 )</td>
<td>41</td>
</tr>
<tr>
<td>Number of Unique Operands</td>
<td>( N_2 )</td>
<td>879</td>
</tr>
<tr>
<td>Total Frequency of Operators</td>
<td>( F_1 )</td>
<td>9876</td>
</tr>
<tr>
<td>Total Frequency of Operands</td>
<td>( F_2 )</td>
<td>12987</td>
</tr>
</tbody>
</table>

| Computed Measures:                        |            |        |
| Program Vocabulary                        | \( N_1 \times N_2 \) | 916    |
| Program Length                            | \( F_1 + F_2 \) | 22863  |
| Estimated Program Length                   | \( N^* = \lceil N_1 \log_2(N_1) + N_2 \log_2(N_2) \rceil \) | 6771.15 |
| Program Volume                            | \( V = (F_1 + F_2) \log_2(N_1 + N_2) \) | 22495.1762 |
| Implementation Level                      | \( L = \text{APPR} \cdot \lceil 2(N_2) \lceil \left\lceil N_1 \lceil F_2 \right\lceil \right\lceil \) | 0.0033 |
| Potential Volume                          | \( V^* = \lceil L \times V \rceil \) | 742.3472 |
| Effort Metric                             | \( EC = V/L \) | 68495960.2303 |
| Programming Time                          | \( T = EC/10 \) | 3802253.3461 sec. |
| Language Level                            | \( \lceil L \lceil L \lceil V \rceil \rceil \) | 2.4497 |
FREQUENCY OF OPERATORS IN DESCENDING ORDER:

1422  I
1501  TO
1433  MOVE
1110  =
778   PERFORM
689   IF
552   ELSE
354   ON
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    WRITE
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.
<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEASURED PROPERTIES:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMBER OF UNIQUE OPERATORS</td>
<td>N1</td>
<td>41</td>
</tr>
<tr>
<td>NUMBER OF UNIQUE OPERANDS</td>
<td>N2</td>
<td>870</td>
</tr>
<tr>
<td>TOTAL FREQUENCY OF OPERATORS</td>
<td>F1</td>
<td>9942</td>
</tr>
<tr>
<td>TOTAL FREQUENCY OF OPERANDS</td>
<td>F2</td>
<td>13200</td>
</tr>
<tr>
<td>COMPUTED MEASURES:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROGRAM VOCABULARY</td>
<td>N1+N2</td>
<td>911</td>
</tr>
<tr>
<td>PROGRAM LENGTH</td>
<td>F1+F2</td>
<td>23142</td>
</tr>
<tr>
<td>ESTIMATED PROGRAM LENGTH</td>
<td>N* = (N1)LOG2(N1)+N2(LOG2(N2)</td>
<td>6715.09</td>
</tr>
<tr>
<td>PROGRAM VOLUME</td>
<td>V = (F1+F2)LOG2(N1+N2)</td>
<td>227510.122</td>
</tr>
<tr>
<td>IMPLEMENTATION LEVEL</td>
<td>L = APPR. (2(N2)/N(N1)(F2)</td>
<td>.0032</td>
</tr>
<tr>
<td>POTENTIAL VOLUME</td>
<td>V* = (L1+V)</td>
<td>728.0515</td>
</tr>
<tr>
<td>EFFORT METRIC</td>
<td>EC = V/L</td>
<td>70763556.2770</td>
</tr>
<tr>
<td>PROGRAMMING TIME</td>
<td>T = EC/16</td>
<td>3931408.6821 SEC.</td>
</tr>
<tr>
<td>LANGUAGE LEVEL</td>
<td>[L1][L][V]</td>
<td>2.3297</td>
</tr>
</tbody>
</table>
FREQUENCY OF OPERATORS IN DESCENDING ORDER:

1691  (  
1500  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
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.

- N-1 -
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.

- 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.
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.