

REFERENCE ONLY



**NUSC Technical Memorandum 841114
14 June 1984**

Computation of the Control Flow Complexity of FORTRAN Modules

**William A. Babson
Marvin J. Goldstein
Computer and Information Services Department**

REFERENCE ONLY



**Naval Underwater Systems Center
Newport, Rhode Island / New London, Connecticut**

Approved for public release; distribution unlimited

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE 14 JUN 1984		2. REPORT TYPE Technical Memo		3. DATES COVERED 14-06-1984 to 14-06-1984	
4. TITLE AND SUBTITLE Computation of the Control Flow Complexity of FORTRAN Modules				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) William Babson; Marvin Goldstein				5d. PROJECT NUMBER 771Y00	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Underwater Systems Center, New London, CT, 06320				8. PERFORMING ORGANIZATION REPORT NUMBER TM 841114	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES NUWC2015					
14. ABSTRACT A Pascal program to compute McCabe's control flow complexity metric for FORTRAN modules is presented. McCabe's metric which is called the cyclomatic number gives the size of any basis set of control flow paths through a program module. McCabe's metric is a useful indicator of the level of difficulty required to test and maintain a program module.					
15. SUBJECT TERMS Pascal; McCabe's metric; Fortran; Special Projects and Studies; cyclomatic number					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 16	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

cy 001

NAVAL UNDERWATER SYSTEMS CENTER
NEW LONDON LABORATORY
NEW LONDON, CONNECTICUT 06320

Technical Memorandum

COMPUTATION OF THE CONTROL FLOW COMPLEXITY OF FORTRAN MODULES

Date: 14 June 1984

Prepared by:

William A. Babson, Jr.
William A. Babson
Computer and Information
Services Department

Marvin J. Goldstein
Marvin J. Goldstein
Computer and Information
Services Department

Distribution Statement "A"
Approved for public release;
distribution unlimited

ABSTRACT

A Pascal program to compute McCabe's control flow complexity metric for FORTRAN modules is presented. McCabe's metric which is called the cyclomatic number gives the size of any basis set of control flow paths through a program module. McCabe's metric is a useful indicator of the level of difficulty required to test and maintain a program module.

ADMINISTRATIVE INFORMATION

This memorandum was prepared under Job Order No. 771Y00, Special Projects and Studies. The authors are located at the Naval Underwater Systems Center, New London, Connecticut, 06320.

INTRODUCTION

McCabe's cyclomatic number (denoted $v(G)$) is a control flow complexity measure derived from graph theory by Thomas J. McCabe [1]. It measures the size of the smallest set of paths which will generate every possible path through a program module. This set may be thought of as a basis set (of control flow paths) for the module.

The cyclomatic number has been proposed for several uses, among them determining the minimum number of test cases required to test every statement in a program and as a method of controlling module size. McCabe suggests that a $v(G)$ greater than ten may lead to testing and maintenance problems.

The program presented to compute McCabe's cyclomatic number is written in Pascal and computes the cyclomatic number as a function of the decision nodes in a program module. A user settable switch is provided to allow the optional viewing of the decision nodes as a module analysis aid. The program will handle any number of FORTRAN modules (main programs, subroutines, functions) in the file it examines as long as each has it's own "END" statement.

BACKGROUND

McCabe's cyclomatic number is a program flow complexity metric taken from Graph theory by defining a mapping between a program module and a directed graph classically called the program control graph. Blocks of sequential code are mapped into the nodes, and transfers of control between blocks into the arcs of the graph.

The cyclomatic number ($v(G)$) of any graph G with e edges, n nodes and p connected components (in this case, the number of connected components can be assumed to be one) is defined to be:

$$v(G) = e - n + 2p \quad (1)$$

The cyclomatic number is the number of linearly independent paths through the graph, which when taken in combination generate all possible paths through the graph. So, by the mapping, the maximum number of linearly independent paths through the program module is given by the cyclomatic number. In his paper, McCabe shows that $v(G)$ may also be computed by:

$$v(G) = d + 1 \quad (2)$$

where d is the sum of one less than the number of outgoing arcs from each decision node. (A decision node is a node that has two or more outgoing arcs.)

Figure 1

```

FUNCTION SIMPSN(FUNC,XMIN,XMAX,N)
C
C   ...THIS FUNCTION INTEGRATES A FUNCTION BY SIMPSON'S
C   ...RULE; FUNC IS THE NAME FOR THE DUMMY FUNCTION TO
C   ...BE INTEGRATED
C
H=(XMAX-XMIN)/N
SUM=0.0
X=XMIN+H
DO 10 I=2,N
IF (MOD(I,2).EQ.0) THEN
    SUM=SUM+4.*FUNC(X)
ELSE
    SUM=SUM+2.*FUNC(X)
ENDIF
X=X+H
10 CONTINUE
SIMPSN=H/3.*(FUNC(XMIN)+SUM+FUNC(XMAX))
RETURN
END

```

In figure 1 a simple FORTRAN module is presented. In figure 2 this module is broken into blocks that represent nodes in graphical form. Figure 3 shows the equivalent program flow graph.

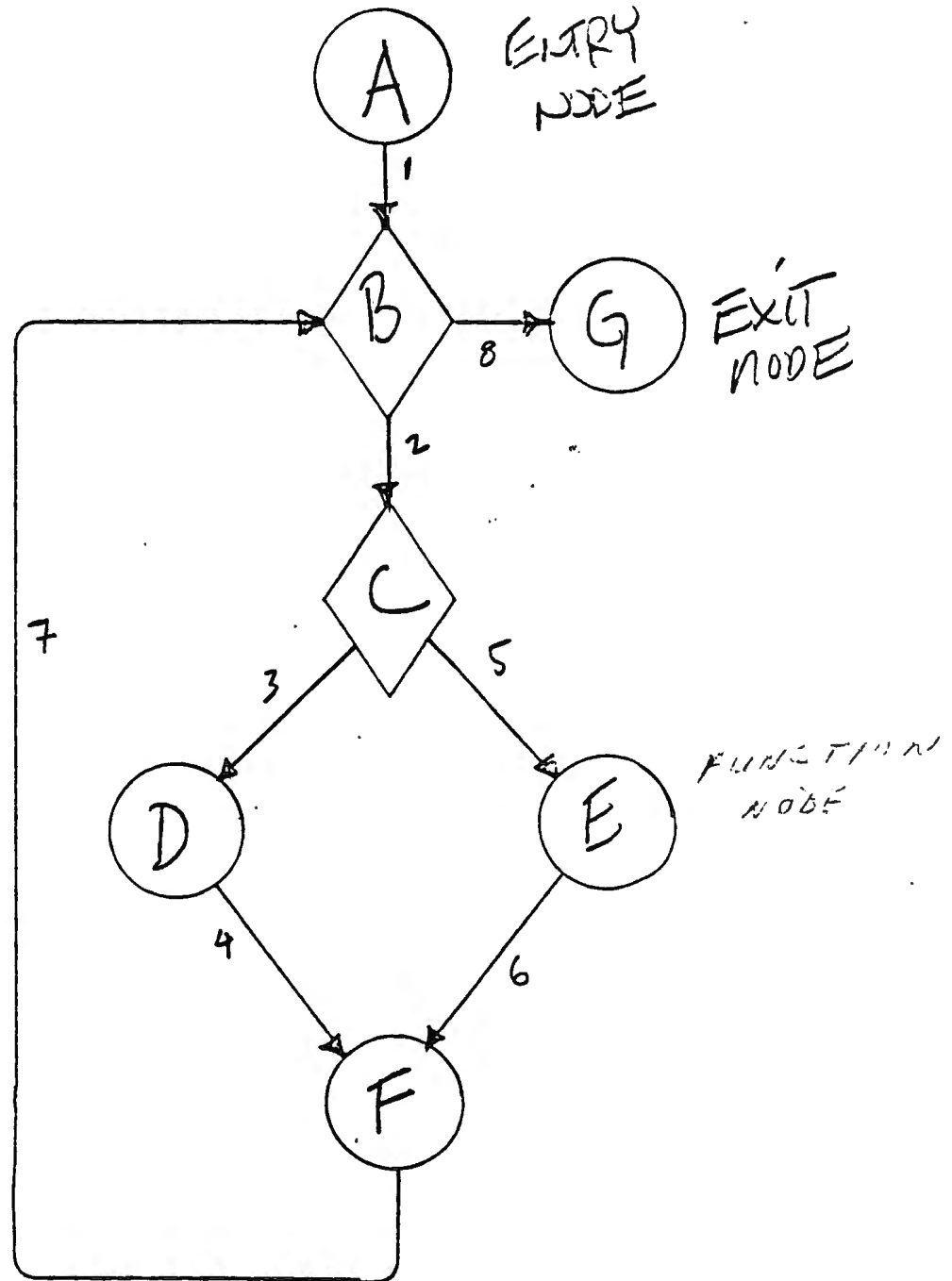
Figure 2

```

FUNCTION SIMPSN(FUNC,XMIN,XMAX,N)
C
C   ...THIS FUNCTION INTEGRATES A FUNCTION BY SIMPSON'S
C   ...RULE; FUNC IS THE NAME FOR THE DUMMY FUNCTION TO
C   ...BE INTEGRATED
C
C-----| -node A -----
C               V
H=(XMAX-XMIN)/N
SUM=0.0
X=XMIN+H
C-----| -node B -(loop(decision))-----
C               V
DO 10 I=2,N
C-----| -node C -(decision)-----
C               V
IF (MOD(I,2).EQ.0) THEN
C-----| -node D -----
C               V
SUM=SUM+4.*FUNC(X)
C-----| -node E -----
C               V
ELSE
SUM=SUM+2.*FUNC(X)
ENDIF
C-----| -node F -----
C               V
X=X+H
10 CONTINUE
C-----| -node G -----
C               V
SIMPSN=H/3.*(FUNC(XMIN)+SUM+FUNC(XMAX))
RETURN
END
C-----

```

Figure 3



By counting the number of nodes and edges and applying (1), or equivalently counting the number of decision nodes and applying (2), the cyclomatic number of the FORTRAN program module may be obtained. In the example above:

$$v(G) = e - n + 2p = 8 - 7 + 2 = 3$$

or

$$v(G) = d + 1 = 2 + 1 = 3$$

In particular, each of the following sets of paths is a basis for the graph in figure 3:

$$\{B C D F B, B C E F B, A B G\} \quad (3)$$

$$\{A B C D F B G, A B C E F B G, A B G\} \quad (4)$$

where:

$$B C D F B = A B C D F B G - A B G$$

$$B C E F B = A B C E F B G - A B G$$

Some properties of $v(G)$ of interest follow:

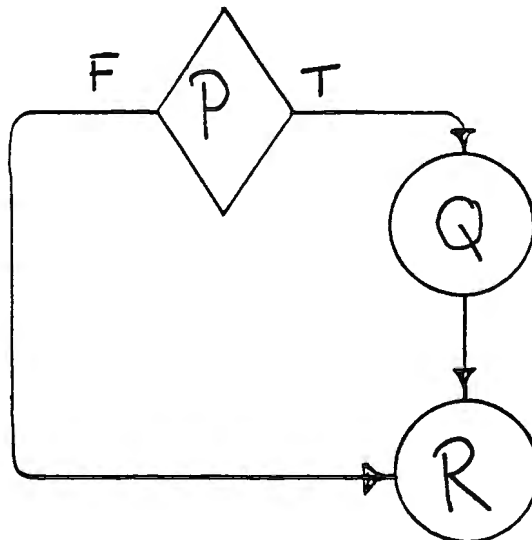
1. $v(G)$ depends only on the decision structure of G .
2. Adding or deleting function nodes in the module does not change $v(G)$.
3. The total cyclomatic number of a program is equal to the sum of the cyclomatic numbers of the modules.

APPLICATIONS

The cyclomatic number bounds the minimum number of test cases that exercise all the statements in a program module. Since a program may have an infinite number of paths through it (an example is DO WHILE (Niagara Falls)), it is not always possible to check every path. However, it is possible to check every statement. Since in the traversal of a maximal set of linearly independent paths every node of the program control graph is visited, the cyclomatic number is an upper bound on the minimum number of tests required to visit all statements in a program module. For example, we draw the reader's attention to the basis set (4) for Figure 3, where only two of the three paths have to be traversed to visit every statement in the program. On the other hand, traversing all paths in a basis set guarantees that every arc in the program control graph is also traversed. Thus, if one can identify a basis of $v(G)$ paths (like (4)) that traverse the program control graph from entry node to exit node, and select a corresponding set of $v(G)$ tests that visit these paths, then one can exercise every transfer of control between program blocks, as well as every statement in the program.

It should be pointed out that having each and every transfer of control traversed at least once is more stringent than having each and every statement executed at least once. This is so because visiting each and every transfer of control implies that every statement in the module is executed, but not conversely. This is illustrated in Figure 4, where traversal of the path PQR executes every statement in the program but fails to execute the branch (P,R).

Figure 4



However, it should be emphasized that the Pascal program described in the next two sections computes only the cyclomatic number. Developing a program that identifies a basis set of paths is a matter that requires further study. In any event, the cyclomatic number can be used to improve a program module's testability - if a module's $v(G)$ is greater than some threshold value, then it should be redesigned to drive its $v(G)$ below the threshold, so that a smaller basis set of tests will visit every statement in the program. Although McCabe suggests a threshold value of 10, this value is by no means sacrosanct.

ALGORITHM

The Pascal program computes the cyclomatic number by operating on all the decision nodes in a file until it encounters a FORTRAN "END" statement. Each decision node is converted into a positive integer that is accumulated in a running sum. When an "END" statement is detected, the number one is added to the sum and then the result is displayed as the cyclomatic complexity count for the module. The running sum is then set to zero and the process repeated until the end of the file is reached.

Specific decision nodes are found by ignoring FORTRAN comments and building complete FORTRAN statements from the remaining source. When a FORTRAN statement is completed, it is examined for keywords. If the statement is identified as a decision node, then either the number one or one less than the number of exits from the decision node is added to the running sum. Otherwise, the statement is discarded and the next statement is constructed and examined. If an "END" statement is detected, the cyclomatic number is computed and displayed. When an end of file condition is detected, the program ends.

Because the program examines statements in this manner, there is no limit on the number of lines in the FORTRAN source file or on the number of modules in the file. The maximum number of decision nodes that it will count in any one module is limited only by the respective (UNIVAC or VAX) Pascal's maximum integer (MAXINT). The program accepts ANSI-77 FORTRAN and recognizes the decision nodes shown in Figure 5.

The Pascal program will not process FORTRAN modules that contain tab characters. Tab characters may be removed by using the CLEAN utility program [4].

Figure 5

The Pascal program recognizes the following FORTRAN control structures and assigns each a complexity count of 1:

- 1) { DO loops }
DO <label> <integer variable> = <field>,<field>
- 2) { Block DO loops }
DO <integer variable> = <field>,<field>
.
.
.
END DO
- 3) { DO WHILE loops }
DO <label> WHILE (<boolean expression>)
- 4) { Block DO WHILE loops }
DO WHILE (<boolean expression>)
.
.
.
END DO
- 5) { Logical IF }
IF (<boolean expression>) <FORTRAN key word>
- 6) { Logical IF in ELSE clause }
ELSE IF (<boolean expression>) <FORTRAN key word>

The Pascal program assigns the complexity number of one less than the number of unique statement labels in the associated label list of each of the following control structures:

- 7) { Arithmetic IF }
IF(<variable>)<label1>,<label2>,<label3>
- 8) { Arithmetic IF in ELSE clause }
ELSE IF(<variable>)<label1>,<label2>,<label3>
- 9) { Computed GOTO }
GOTO(<label1>,...,<labeln>),<integer variable>
- 10) { Computed GOTO in ELSE clause }
ELSE GOTO(<label1>,...,<labeln>),<integer variable>

The Pascal program allows Computed GOTO's and Arithmetic IF statements to follow Logical IF's (or ELSE IF's) as per ANSI_77[5]

- 11) { Arithmetic IF following Logical IF }
[ELSE] IF(<boolean expression>) IF(<variable>)
+ <label1>,<label2>,<label3>
- 12) { Computed GOTO following Logical IF }
[ELSE] IF(<boolean expression>) GOTO(<label1>,...,<labeln>),
+ <integer variable>

USING THE PROGRAM

In the two examples below, everything that the user would type is underlined.

I. Using the Program on the VAX (V701, V703, V70A)

To compute the cyclomatic complexity of the program in figure 1 on the VAX the following DCL may be used, where CYCLTEST.FOR is the file name of the program in figure 1.

```
$ ASSIGN/USER MODE CYCLETEST.FOR INFILE
$ RUN [MJG.CLEAN]CYCL
```

print out decision nodes ? <y,n>: Y

```
DO 10 I=2,N
  IF (MOD(I,2).EQ.0)THEN
```

McCabe's cyclomatic number for module 1 is 3.

<ENDFILE>

The utility is not case sensitive. If the Tab-Formatting feature available with the VAX FORTRAN has been used, the CLEAN utility will have to be used before invoking the cyclomatic number program [4].

II. Using the Program on the UNIVAC:

On the UNIVAC, the cyclomatic number program may be found in CYCL*PAS.UTIL on node 2 in New London. For the following sample runstream an imaginary FORTRAN source element of FOO*BAR.CYCLTEST which holds the program given in Figure 1 will be used.

>ASG,A FOO*BAR.

>READY

>ASG,T INFILE.

>READY

>ED FOO*BAR.CYCLTEST.INFILE.

>XQT CYCL*PAS.UTIL

>

> print out decision nodes ? <y,n>: Y

>

> DO 10 I=2,N

IF (MOD(I,2).EQ.0)THEN

>

> McCabe's cyclomatic number for module 1 is 3.

>

> <ENDFILE>

>

CONCLUSION

A Pascal program is presented which computes McCabe's cyclomatic number, the cardinality of any basis set of control flow paths through a program module. Since all paths through the module may be generated from a basis set, the cyclomatic number bounds the minimum number of tests needed to exercise every transfer of control between program blocks in the program module, as well as every program statement. Therefore, the Pascal program may be used to earmark program modules for redesign that require a large number of tests to validate the correctness of every transfer of control. The redesign effort should strive to reduce the cyclomatic number of any program module to some acceptable threshold value in order to improve the module's maintainability with respect to its testability.

REFERENCES

1. Thomas J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Volume SE-2, number 4, (December, 1976).
2. Glenford J. Meyers, "An Extension to the Cyclomatic Measure of Program Complexity," ACM SIGPLAN NOTICES, Volume 12 (October 1977).
3. Wilfred J. Hansen, "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator (Count)," ACM SIGPLAN, Volume 13 (May 1978).
4. M. J. Goldstein and John Lawson, Jr., "A New Program Aid in Producing Structured FORTRAN Programs," NUSC TM No. 821162 (November, 1982).
5. American National Standards Programming Language FORTRAN ANSI X3.9 1977,
American National Standards Institute, New York, NY (1978).
6. M. J. Goldstein and John Lawson, Jr., "An Example of Quality Mathematical Software," NUSC TM No. 811044, (15 April, 1981).
7. Brian W. Kernighan and P. J. Plauger, The Elements of Programming Style,
McGraw-Hill Book Company (1974).

COMPUTATION OF THE CONTROL FLOW COMPLEXITY OF FORTRAN MODULES

William A. Babson

Marvin J. Goldstein, Computer and Information Services Dept.

TM No. 841114

14 June 1984

UNCLASSIFIED

DISTRIBUTION LIST

EXTERNAL

Mr. I. L. Avrunin
David Taylor Naval Ship
Research & Development
Center
Headquarters
Bethesda, MD 20084

Mr. James Dooley
Naval Data Automation
Command
Washington Navy Yard
Washington, DC 20374

Mr. John Baird
NOSC
San Diego, CA 92152

Ms. Mary Ann Engelbert
Naval Data Automation
Command
Washington Navy Yard
Washington, DC 20374

Ms. Fran Kazlauskis
Naval Data Automation
Command
Washington Navy Yard
Washington, DC 20374

Dr. Martin Schultz
Research Center for Scientific Comp.
Yale Univ.
P. O. Box 2158 Yale Station
New Haven, CT 06520

Mr. Ken Kunec
Naval Data Automation
Command
Washington Navy Yard
Washington, DC 20374

Mr. Al Poulin
Naval Intelligence Command
Headquarters
4600 Silver Hill Road
Washington, DC 20389

Ms. Becky Swales
Naval Data Automation
Command
Headquarters
4600 Silver Hill Road
Washington, DC 20389

Mr. Larry Williams
Naval Intelligence
Command
Headquarters
4600 Silver Hill Road
Washington, DC 20389

Internal:

Code 00 CAPT Ailes
 01 E. L. Messere
 01A John Keegan
 039 Dr. D. M. Viccone
 10 Dr. W. A. VonWinkle
 101 Dr. E. Eby
 L. Goodman
 02 D. C. Kindilien
 0211 R. H. Bernier
 02 NL Library (3 copies)
 021311 NPT Library

 20 W. L. Clearwaters
 234 R. J. Conley
 32 A. Lesick
 Dr. N. L. Owsley
 Dr. John Ianniello
 J. K. Sullivan
 R. Streit
 W. Axtell
 R. C. Cox
 James Pearson
 L. C. Ng
 J. V. Sanchis
 J. Ferrie
 B. Helme, Jr.
 Dr. C. H. Sherman
 W. Fox
 B. G. Buehler
 A. W. Ellenthorpe
 Dr. Peter G. Cable
 Dr. S. Ko
 D. T. Porter
 W. H. Wharton
 T. Anderson
 J. Ionata
 M. Kuuznitz
 J. Gannon
 C. Bowman
 P. R. Miner
 Rosemany Molino
 Jose Munoz
 D. Rawson
 J. Shores
 W. Strawderman
 R. Leask
 D. Yarger
 Russell Christman

 33 Dr. A. H. Nuttal
 J. B. Paniszczy
 Dr. F. R. DiNapoli
 E. P. Jenson
 E. G. Kanabis

Code 33 S. C. Gerengher
 J. Gregor
 W. A. Goldman
 P. M. Anchors
 J. J. Higgs
 I. B. Cohen
 D. W. Counsellor
 Dr. R. Radliniski
 D. A. Fingerman
 R. J. Trembley
 J. E. Miller
 Dr. H. F. Dwyer
 Dr. C. Carter
 A. Quazi
 Peter Stahl
 J. Wolcin
 Dr. M. B. Moffett
 C. J. Becker
 E. C. Gannon
 R. J. MacDonald
 J. Saikowski
 G. Botseas
 Dr. D. Lee
 J. Nordquist
 E. R. Robinson
 H. Weinberg
 H. Sternberg
 Dr. David Wood
 Michael Fecher

 34 J. R. Katan
 Dr. D. E. Fessenden
 K. F. Hafner
 John Casey
 Anthony Bruno
 D. Dixon

 35 Tom Conrad
 L. Cabral

 36 R. P. Eidimtas
 H. A. Rosen
 R. V. Cherry
 W. J. Ryan
 J. Griffin
 D. G. Blundell
 M. P. Lydon
 S. I. Wax
 S. E. Ashton
 Raymond McMahon
 S. Meyers

 37 C. M. Curtis
 Q. Huynh

 38 J. E. Sims
 W. H. Greene

 401 A. D. Carlson

Internal:

Code 401	J. H. Clark	Code 73	R. Forget
	Dr. A. J. Kalinowski		A. Sullivan
	Dr. R. G. Kasper		J. Surdo
	R. R. Manstan	74	M. Lee
	E. L. McLaughlin		C. Brockway
	R. S. Munn		M. Becker
	C. W. Nebelung		L. Doyle
	Dr. J. S. Patel		J. Koonce
	B. A. Radley		J. Pappadia
	A. Y. Shigematsu		A. Levander
	M. A. Tuccio		Norman Dube Npt
402	M. Berger		A. Blau Npt
4111	Samuel Horvitz		D. Stanhope Npt
4331	B. L. Antrim		R. Hoy Npt
	S. B. Walsh		N. Bradbury Npt
434	K. Steele		
	G. Lussier		
5202	Stephan Schady		
60	Dr. D. Bordelon		
	Dr. Jeffrey Cohen		
70	Cort R. DeVoe		
701	R. Wilson Npt		
	D. McCue		
	J. Babiec Npt		
71	G. Elias Npt		
	M. Goldstein (15 Copies)		
	D. Drinkard		
	D. Aker		
	N. Sulinski		
	W. Babson (15 Copies)		
	E. Montavon		
	J. Sikorski		
	Louis Ledoux Npt		
	R. Johnson		
	Cynthia Borges Npt		
	Marcel Nadeau Npt		
	John Ventura Npt		
	Phoebe Liu Npt		
	John Sabulis Npt		
72	Gordon Daglieri Npt		
	S. Schneller Npt		
	J. Auwood		
	T. Wheeler		
	R. Warren Npt		
	P. Breslin Npt		
	A. Alfiero		
	R. Clark		
	J. Gribbin		
73	D. Quigley		
	S. Capizzano		
	J. MacDonald		
	R. Pingree		
	R. Cote		