AD-A035 778 TRW INC REDONDO BEACH CALIF F/G 9/2
FORTRAN CODE AUDITOR. VOLUME I. USER'S MANUAL.(U)
DEC 76 P SMITH F30602-76-C-0187
UNCLASSIFIED RADC-TR-76-395-VOL-1 NL

1 OF 1
ADA035778

END

DATE
FILMED
3-77

ADA035778

RADC-TR-76-395, Volume I (of two)
Final Technical Report
December 1976

FORTRAN CODE AUDITOR
User's Manual

TRW

COPY AVAILABLE TO DDC DOES NOT
PERMIT FULLY LEGIBLE PRODUCTION

Approved for public release;
distribution unlimited.

DDC
RECEIVED
FEB 22 1977
A

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

This report contains a large percentage of machine-produced copy which is not of the highest printing quality but because of economical consideration, it was determined in the best interest of the government that they be used in this publication.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: Donald L. Mark

DONALD L. MARK
Project Engineer

APPROVED: Robert D. Kreutz

ROBERT D. KREUTZ, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and adva
development programs in command, control, and communic
(C^3) activities, and in the C^3 areas of information sc
and intelligence. The principal technical mission ar
are communications, electromagnetic guidance and cont
surveillance of ground and aerospace objects, intellig
data collection and handling, information system tech
ionospheric propagation, solid state sciences, electro
physics and electronic reliability, maintainability a
compatibility.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-76-395, Vol I (of two)	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): FORTRAN CODE AUDITOR. Volume I. User's Manual.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. May 1975 - January 1976.
6. PERFORMING ORG. REPORT NUMBER: N/A	
7. AUTHOR(s): Paul Smith	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0187
9. PERFORMING ORGANIZATION NAME AND ADDRESS: TRW Inc., One Space Park, Redondo Beach CA 90278	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 55811409 62702F
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441	12. REPORT DATE: December 1976
	13. NUMBER OF PAGES: 56
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Donald L. Mark (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

FORTRAN Coding Standard
Structural Analysis
Conventions Audit
Automatic Test Tool

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The FORTRAN Code Auditor, an automated test tool, is used for the cost effective enforcement of FORTRAN programming standards and conventions appropriate to the Air Force software environment. It does not modify code. Using predefined coding standards and conventions, it simply advises the user where these standards and conventions have not been adhered to. The major advantage of favoring an automated auditor over manual methods, besides cost effectiveness, is complete objectivity and unambiguity. (over) next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

390 311

LB

cont.

The standards can be viewed as being coding enforcements in four areas:

1. Documentation Standards - Standards defining quantity and placement of commentary thus enhancing program readability and comprehensive;
2. Format Standards - Standards identifying physical placement and grouping of code elements on the source code listing;
3. Design Standards - Standards limiting module size and placing restrictions on the use of certain instructions with the end result of providing an optimization of code relative to execution time; and
4. Structural Standards - Standards requiring the use of strict rules for the top-down design and implementation of a system of programs and the requirement that the components adhere to a hierarchical form as much as possible.

EVALUATION

The feasibility and value of code auditing to enforce programming standards have been successfully demonstrated. Most major software development projects have some form of programming standards, but rarely are they enforced, and then not effectively. Such standards generally address four areas:

* Documentation - the placement, orientation, and amount of comments in a program
* Format - the placement and grouping of code elements
* Design - the use of language constructs, program structure, and module size
* Structural - use of top-down design and implementation

The purpose of programming standards is to make program modules, coded by various people, more uniform, readable, and understandable; to make them easier to debug, test, and maintain; and to restrict the use of coding practices which experience has shown to be prone to error. Especially, programming standards define what is good programming practices.

RADC's Code Auditor, an automatic test tool, will be used for the cost effective enforcement of FORTRAN programming standards and conventions appropriate to the Air Force software environment. The single biggest advantage of favoring an automated auditor over manual methods, besides its cost effectiveness, is its complete objectiveness and unambiguity.

Donald L. Mark

DONALD L. MARK
Project Engineer

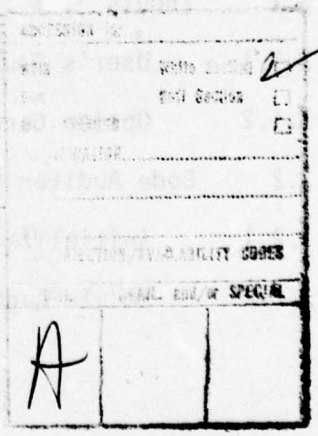


TABLE OF CONTENTS

SECTION		PAGE
1.0	GENERAL DESCRIPTION	1
1.1	Purpose of the User's Manual	1
1.2	Project Reference	2
2.0	SYSTEM SUMMARY	3
2.1	System Application	3
2.2	System Operation	3
2.3	System Configuration	5
2.4	System Organization	5
2.5	Performance	8
2.5.1	Input Limits	8
2.5.2	Output Limits	8
2.5.3	Processing Time	8
2.5.4	Other Limits	8
3.0	GENERAL DESCRIPTION OF INPUTS, PROCESSING AND OUTPUTS	10
3.1	Inputs	10
3.1.1	User's Source Input	10
3.1.2	Option Card Input	10
3.2	Code Auditor Outputs	13
3.2.1	Initialization Output (Coding Standards Audit)	13
3.2.2	Module Audit Output (Coding Standards Audit)	13

SECTION		PAGE
3.2.3	Module Summary Output (Coding Standards Audit) . . .	15
3.2.4	Program Summary Output (Coding Standards Audit) . .	18
3.2.5	Module Segmentation Output (Program Structural Analysis)	20
3.2.6	Segment Transfer Table Output (Program Structural Analysis)	21
3.2.7	Module Summary Output (Program Structural Analysis)	24
3.2.8	Error Messages	27
3.3	Operating Procedures	30
3.3.1	Source Input via Cards	31
3.3.2	Source Input From Disk	34
3.3.3	Source Input From Tape	35
APPENDIX A	RADC FORTRAN Code Auditor Coding Standards	36
APPENDIX B	Structured Program Requirements	40
B-1	Program Segmentation	40
B-2	Segment Transfer Table	43
B-3	Structural Analysis	44

1.0 GENERAL DESCRIPTION

1.1 Purpose of the User's Manual

The purpose of this User's Manual is to describe the use and operation of the RADC FORTRAN Code Auditor. The Code Auditor, an automated test tool, is used for the cost effective enforcement of FORTRAN programming standards and conventions appropriate to the Air Force software environment. It does not modify code. Using pre-defined coding standards and conventions, it simply advises the user where these standards and conventions have not been adhered to. The single biggest advantage favoring an automated auditor over manual methods, besides its cost effectiveness, is its complete objectiveness and unambiguity.

The standards can be viewed as being coding enforcements in four areas:

(1) Documentation Standards - Standards defining quantity and placement of commentary thus enhancing program readability and comprehension.

(2) Format Standards - Standards identifying physical placement and grouping of code elements on the coding sheet, again enhancing readability and comprehension.

(3) Design Standards - Standards limiting module size and placing restrictions on the use of certain instructions with the end result of providing an optimization of code relative to execution time.

(4) Structural Standards - Standards requiring the use of strict rules for the top-down design and implementation of a system of programs and the requirement that the components adhere to a hierarchical form as much as possible.

1.2 Project Reference

(1) Control Cards Reference Manual
Order Number BS19

(2) File Management Supervisor
Order Number DB54

(3) Fortran IV
Order Number BN88

(4) GCOS Time-Sharing Terminal/Batch Interface Facility
Order Number BR99

(5) General Comprehensive Operating Supervisor
Order Number BR43

(6) General Loader
Order Number BN90

(7) Guidelines for Software Quality Assurance
TRW Number SPD-3055
Date 9-75

2.0 SYSTEM SUMMARY

2.1 System Application

The RADC FORTRAN Code Auditor Program provides project personnel with an automatic means of auditing their computer programs to insure that the programming practices contained within their programs conform to RADC Software Coding Standards and Procedures. These programming standards are described in Appendix A. The RADC FORTRAN Code Auditor also monitors the user's source to determine if the code is structured, i.e., consists of combinations of the control structures depicted in Appendix B, such that control flows from top to bottom or from beginning to end.

2.2 System Operation

The RADC FORTRAN Code Auditor is initially used by the programmer during program development and only after successful compilation to identify those areas within his code which do not adhere to the predefined coding standards and conventions. Utilization of the tool during the program development stage allows the programmer to concurrently reduce or eliminate any standards violations prior to unit testing and quality assurance examinations. The latter activity also makes use of the Code Auditor in officially sanctioning the program's adherence to established coding conventions. Figure 2-1 in general, depicts the timing of the application of the Code Auditor during the software development cycle. The Code Auditor has been designed such that its output is clear and unambiguous thus enabling its use by both technical and management personnel.

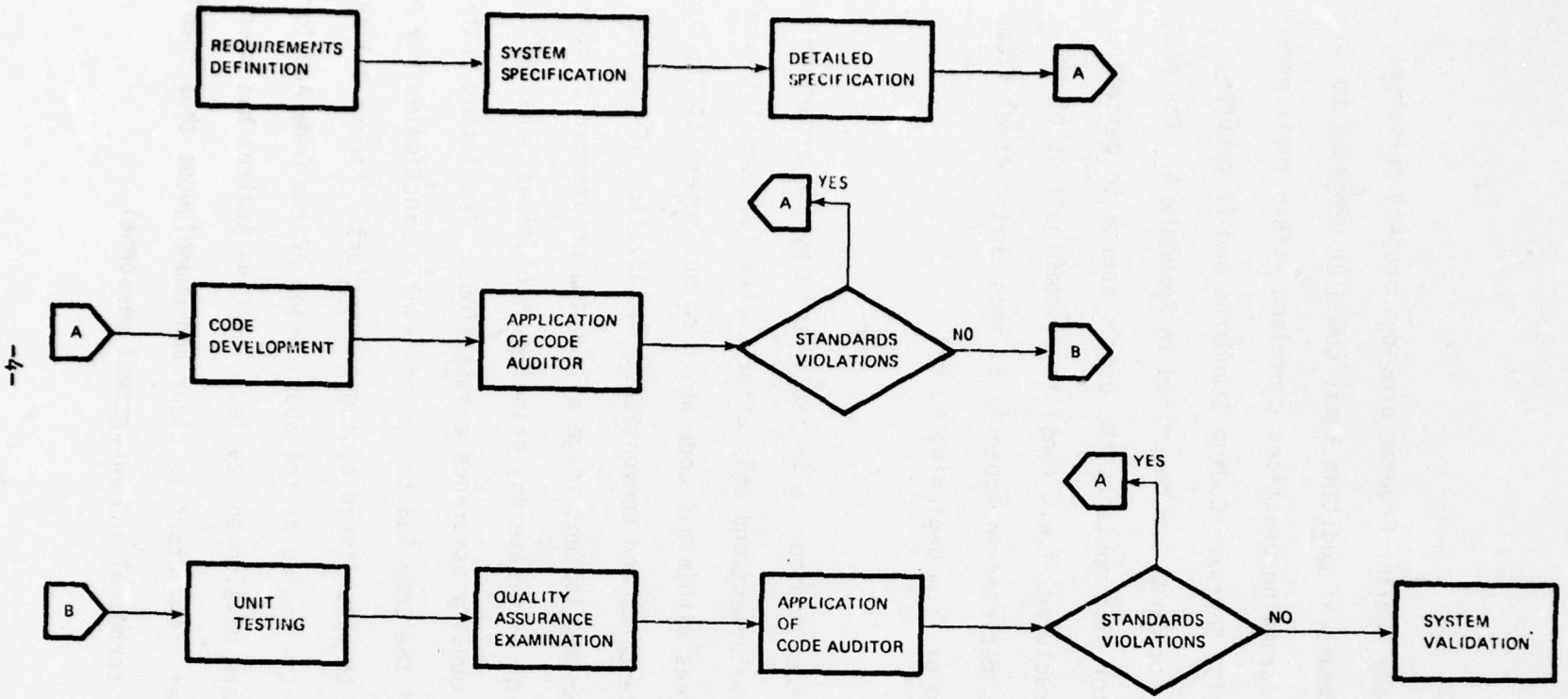

REQUIREMENTS DEFINITION
SYSTEM SPECIFICATION
DETAILED SPECIFICATION
A
A
YES
A
CODE DEVELOPMENT
APPLICATION OF CODE AUDITOR
STANDARDS VIOLATIONS
NO
B
A
YES
B
UNIT TESTING
QUALITY ASSURANCE EXAMINATION
APPLICATION OF CODE AUDITOR
STANDARDS VIOLATIONS
NO
SYSTEM VALIDATION

Figure 2-1

2.3 System Configuration

The minimum configuration required to support the RADC FORTRAN Code Auditor shall include:

1. Honeywell Series 600/6000 Information Processing System under GCOS control
2. 40K words of memory
3. DSS180 Disk Storage Subsystem or the equivalent; a minimum of 1 drive
4. 36 - bit word length
5. Card Reader - Reads punched 80 column cards (CRZ-201 or equivalent)
6. Line Printer - Prints 132 columns per line (PRT 300/PRT 201 or equivalent)
7. Software - FORTRAN Y Compiler (Honeywell Extended Compiler)

2.4 System Organization

The Code Auditor consists of three physical segments (overlays) which are functionally consistent with the three basic logical components of the system and their attendant operation. The operations performed by the Code Auditor's logical parts and their correlation to the appropriate physical system overlays are:

(1) Analyze the user's source code to check for non-adherence to the established standards and conventions except the program's structural requirements, reporting any non-conformance monitored. Overlay A performs this function completely independent of overlays B and C processing.

(2) Overlay B performs a second pass analysis of the user's source code, parsing its structure into segments and relationships among segments, for subsequent processing by overlay C. The Segment Transfer Table, which describes this interrelationship among the program's segments, is written to a temporary disk file and constitutes the single interface between overlays B and C.

(3) Overlay C accepts as input the temporary file created in overlay B containing the Segment Transfer Table. Via iterative application of the reduction algorithm discussed in Appendix B, an analysis is made of the Segment Transfer Table to establish compliance with the structured programming rules. This overlay prints the Segment Transfer Table as passed from overlay B, its contents after iterative reduction (if unstructured), and summarily a "structured/unstructured" message for each module of the user's source modules processed.

Figure 2-2 depicts for each logical segment, its interface with other segments, files processed and reports generated. The main driver (root segment) is interfaced via a labeled common to each of the three overlays comprising the Code Auditor system. Besides providing root segment access to Code Auditor files, the labeled common block passes option card parameters for execution control of the individual overlays.

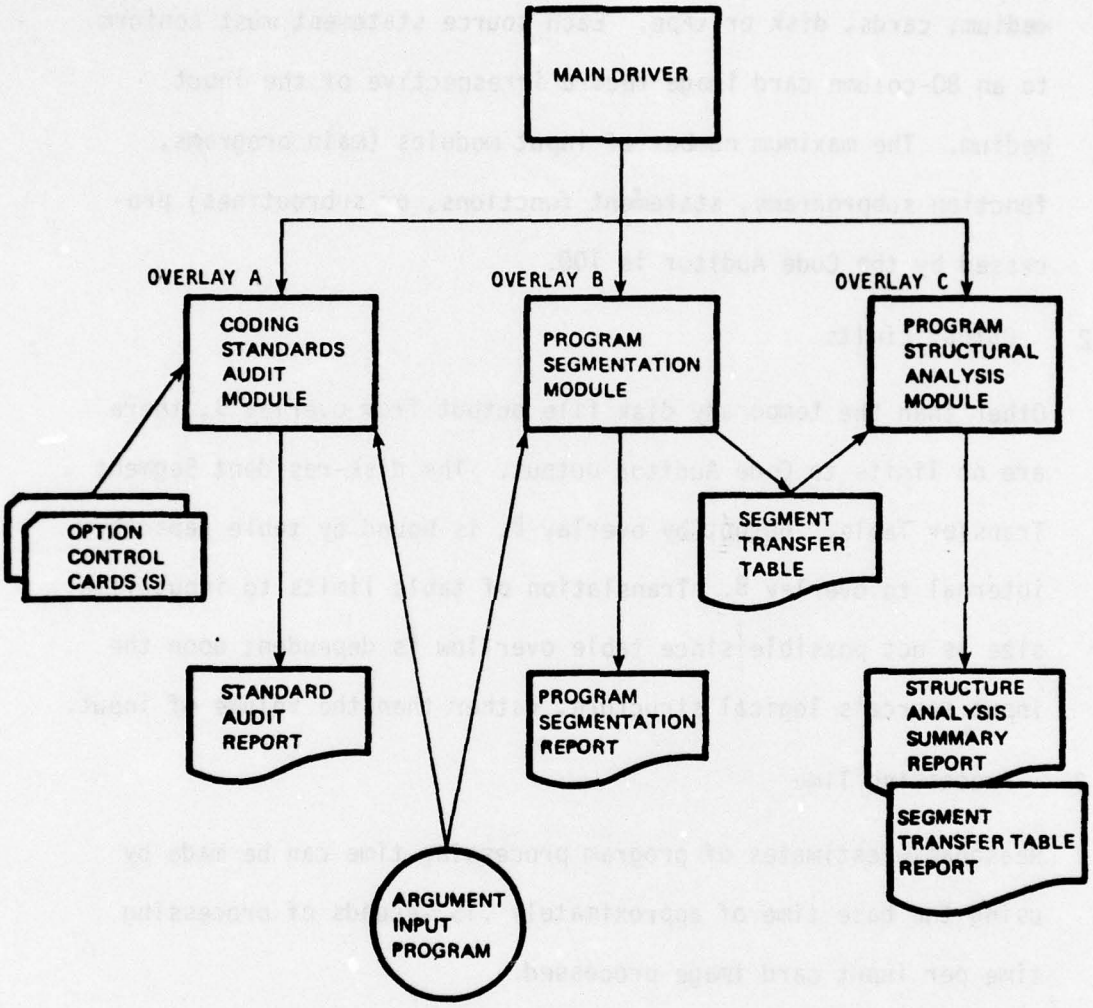


Figure 2.2. Code Auditor Organization and Interfaces

2.5 Performance

2.5.1 Input Limits

The Code Auditor will process user source input from any input medium; cards, disk or tape. Each source statement must conform to an 80-column card image record irrespective of the input medium. The maximum number of input modules (main programs, function subprograms, statement functions, or subroutines) processed by the Code Auditor is 100.

2.5.2 Output Limits

Other than the temporary disk file output from overlay B, there are no limits to Code Auditor output. The disk-resident Segment Transfer Table, output by overlay B, is bound by table capacity internal to overlay B. *Translation* of table limits to input file size is not possible since table overflow is dependent upon the input source's logical structure, rather than the volume of input.

2.5.3 Processing Time

Reasonable estimates of program processing time can be made by using the base time of approximately .15 seconds of processing time per input card image processed.

2.5.4 Other Limits

Additional limitations and constraints to source content follow:

(1) FORTRAN syntax errors are not detected by the Code Auditor. In order for Code Auditor to audit properly, all syntax

errors detectable by the compiler should be absent from the input.

(2) A blank character (hollerith blank) is recognized as a delimiter; therefore, embedded blanks within variable names, keywords, etc., will cause an incorrect scan of the elements in the card image.

(3) When the programmer includes an IMPLICIT statement in the program specifying alphabetic characters as being either INTEGER or DOUBLE PRECISION variable types, and subsequently uses a specification statement redefining the implicity defined variable to something other than DOUBLE PRECISION or INTEGER variable type, the Code Auditor will not determine that standard no. 14 for mixed mode arithmetic or standard no. 22 for I/O device referenced as an integer name has been violated.

(4) When the Code Auditor encounters a comment card imbedded within continuation cards, data on the remaining continuation cards are not analyzed for adherence to the pre-defined coding standards.

(5) Variable or array names contained within labeled common and additionally declared as CHARACTER may cause improper monitoring of standard no. 24; common block length must be the same in all modules.

3.0 GENERAL DESCRIPTION OF INPUTS, PROCESSING AND OUTPUTS

3.1 Inputs

The Code Auditor inputs consist of the user's input source code and one or more option cards.

3.1.1 User's Source Input

The Code Auditor's source input may be input via either of three mediums; cards, disk or tape. Irrespective of the input medium, all source statements must be 80-column card-image format. The source code input may constitute any number of FORTRAN modules, up to 100, delimited in the usual manner by the FORTRAN END Card.

3.1.2 Option Card Input

The option card provides the user with a method of directing the Code Auditor to execute optional or alternate functions of the Code Auditor. It may only be input from the card reader. The optional functions supported are:

(1) List the entire input source code along with Code Auditor assigned card sequence numbers and indicators showing non-adherence to coding standards.

(2) Limit the output listing of the input source code to those card images containing source code where non-adherence occurs along with assigned card numbers and indicators.

(3) Suppress auditing for adherence to any one or more of the coding standards.

(4) Suppress auditing for all of the coding standards, nos. 1 thru 37.

(5) Suppress auditing for adherence to program structural requirements.

(6) The input source is realtime code.

These controls are input on the option card. The option card is identified by the character string "CAOPTION" in card columns 1 through 8. Other parameters may appear in any sequence. When the Code Auditor encounters no CAOPTION card at the start of execution, the Code Auditor will provide default options as follows:

(1) List the entire source input.

(2) Audit for adherence to all non-realtime coding standards.

(3) Audit for adherence to structural programming requirements.

The parameter character strings which control the options and their effect on Code Auditor processing are shown in the list below.

Parameter Character String	Effect on Code Auditor Processing
CAOPTION	Directs Code Auditor to search the card for user controls.
LIST	Directs Code Auditor to list the entire source code input.
NOLIST	Directs Code Auditor to limit the source code listing. (Note: When Code Auditor finds neither the parameter LIST nor NOLIST the default is LIST. When Code Auditor finds both parameters LIST and NOLIST the default is NOLIST.)
REAL	Directs Code Auditor to refrain from auditing for adherence to the coding standard numbers 27 and 28.
NOCA	Directs Code Auditor to refrain from auditing for adherence to coding standard nos. 1 thru 37.
NOSTRUCT	Directs Code Auditor to refrain from auditing for adherence to structural coding requirements.
1	Directs Code Auditor to refrain from auditing for adherence to coding standard number 1.
2	Directs Code Auditor to refrain from auditing for adherence to coding standard number 2. . .
37	Directs Code Auditor to refrain from auditing for adherence to coding standard number 37. (Note: Each of the numbers from 1 to 37 is a control parameter. Each number n input directs Code Auditor to refrain from auditing for adherence to coding standard number n.)

The CAOPTION card is identified to Code Auditor by the string of characters "CAOPTION" in columns 1-8 of the card. Other parameters may appear in any sequence. The parameters are delimited by a string of one or more characters of blanks and/or commas in any sequence.

3.2 Code Auditor Outputs

All of Code Auditor's outputs are printed data. Code Auditor prints during eight distinct phases of its execution; four phases each for coding standards audit and program structural analysis respectively. The first four phases of printed output occur during coding standards audit processing. Phases five through eight occur during program structural analysis processing. Code Auditor outputs are described in sections 3.2.1 through 3.2.7.

3.2.1 Initialization Output (Coding Standards Audit)

There are two initialization messages. The first message reflects Code Auditor's interpretation of options selected by the user. It reports the user's direction to Code Auditor with respect to the options NOLIST and REAL.

The second message is a list of the standards (see figure 3-1).

3.2.2 Module Audit Output (Coding Standards Audit)

The data output during the audit of each module are the listing of the source code being audited, the Code Auditor

```
                    RADC CODE AUDITOR

                    STANDARDS AUDIT

        MODULE ORGANIZATION INCLUDES: CODE AUDITOR
OPTIONS IN EFFECT  LIST , NON REAL TIME, 80 COLUMNS          OPTIONS SELECTED BY USER - MESSAGE 1

SUBSECTION DESCRIPTION
--+-b-------c---d-----e---f---g---h--------

 1    NOT USED
 2    STMT LABELS ARE TO BE IN COLUMNS 2-5,RIGHT JUSTIFIED.
 3    STMT LABELS ARE TO BE IN ASCENDING ORDER.
 4    CONTINUATION STMTS TO BE SEQUENCED: 1-9,A-J.
 5    NOT USED
 6    MAXIMUM OF 100 EXECUTABLE STATEMENTS PER ROUTINE.
 7    NOT USED
 8    USE INTEGER SUBSCRIPTS WHEN REFERENCING AN ARRAY.
 9    DATA STMTS TO PRECEDE EXECUTABLE CODE.
10    ONE TYPE SPECIFICATION FOR A VARIABLE NAME;
11    ARGUMENTS IN CALL STMTS MUST NOT CONTAIN ARITH. OR LOGICAL EXPRESSIONS.
12    REAL TIME ROUTINES MAY NOT HAVE ARGUMENTS IN CALL STMTS;
13    ONE ASSIGNMENT STATEMENT PER LINE.
14    NO MIXED MODE ARITHMETIC ON RIGHT SIDE OF EQUAL SIGN.
15    WHOLE NUMBERS USED AS EXPONENTS MUST BE INTEGER;
16    NOT USED
17    NOT USED
18    DO LOOP NESTS MUST NOT EXCEED SIX LEVELS.
19    FORMATS ARE PLACED AFTER I/O STATEMENTS, OR AFTER RETURN ;
20    NOT USED
21    NOT USED
22    I/O DEVICES MUST BE REFERRED TO BY INTEGER VARIABLE NAME.
23    COMPUTED GO TO STMTS ALLOWED IF GO TO VARIABLE IS CHECKED.
24    COMMON BLOCK LENGTH MUST BE THE SAME IN ALL MODULES.
25    ALL COMMON BLOCKS WILL BE LABELED.
26    NEVER OMIT ARRAY SUBSCRIPTS EXCEPT IN DATA,CALL OR OUTPUT STMTS;
27    STOP AND PAUSE STMTS NOT ALLOWED IN REAL-TIME PROGRAMS.
28    ASSIGNED GO TO STMTS ARE NOT ALLOWED IN REAL-TIME PROGRAMS.
29    NOT USED
30    NOT USED
31    PREFACE COMMENTARY BLOCK TO CONTAIN:
        C OR * IN COLUMN 1 ON ALL CARDS
        FIRST CARD:          * IN COL 2 OR 3 THRU 71 OR 72
        INTERMEDIATE CARDS:  SAME SYMBOL IN COL 1 AS FIRST CARD
        LAST CARD:           * IN SAME COLUMNS AS CARD 1
32A   COMMENTS MUST PRECEDE BLOCKS OF ONE OR MORE IF STMTS.
32B   COMMENTS MUST PRECEDE BLOCKS OF ONE OR MORE CALL STMTS.
32C   COMMENTS MUST PRECEDE BLOCKS OF ONE OR MORE I/O STMTS.
32D   COMMENTS MUST PRECEDE MIXED MODE ASSIGNMENT STMTS.
33    IN-LINE COMMENTS CONSIST OF THREE OR MORE CARDS:
        C OR * IN COLUMN 1 ON ALL CARDS
        FIRST CARD:          BLANKS IN COLS 2-72
        INTERMEDIATE CARDS:  TEXT
        LAST CARD:           BLANKS IN COLS 2-72
34    NO RETURN STMT CAN CONTAIN AN ARGUMENT LIST;
35    NOT USED
36    NOT USED
37    NOT USED
```

LIST OF STANDARDS - MESSAGE 2

FIGURE 3-1

-14-

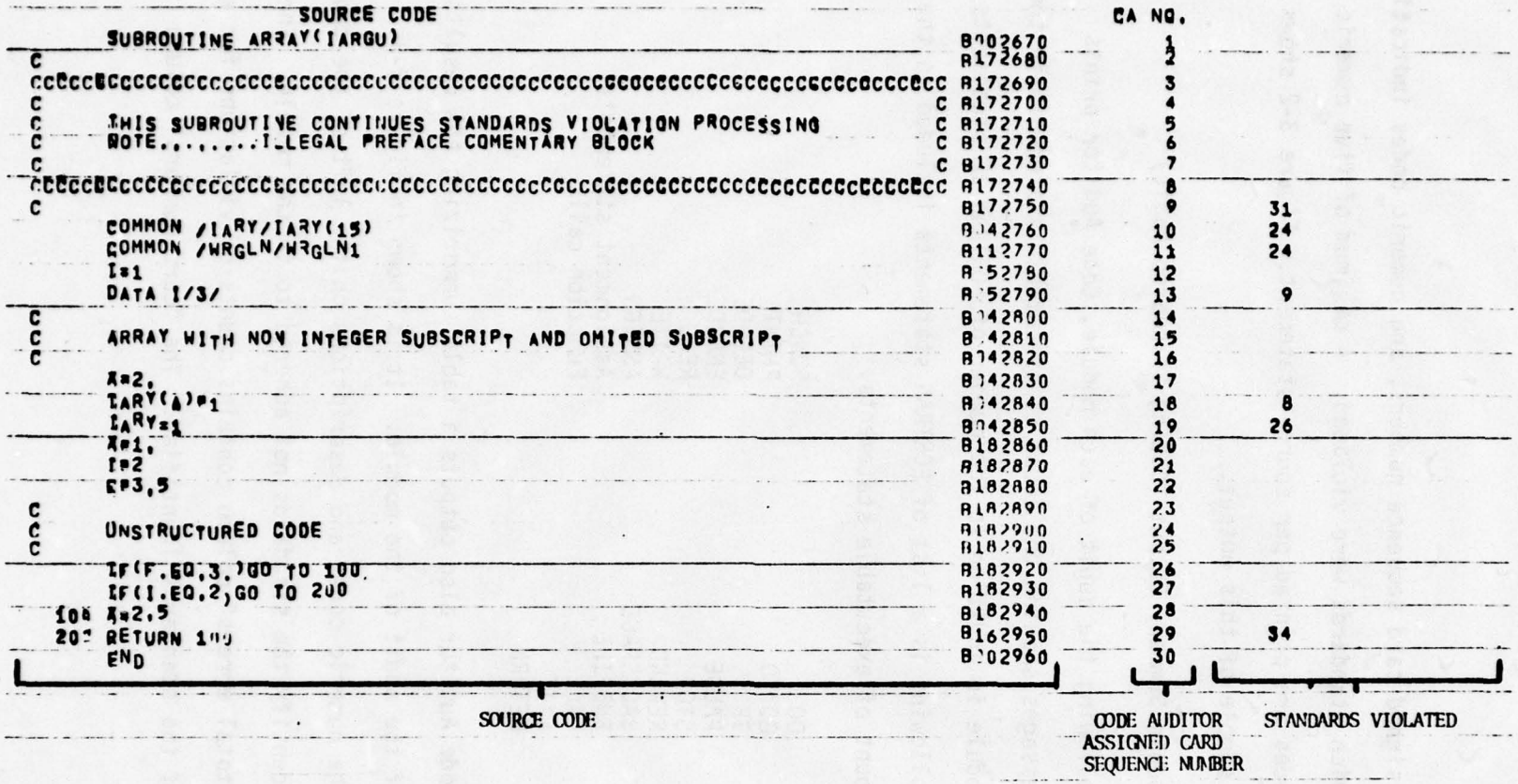
assigned card sequence numbers, and numeric codes indicating which standards were violated. A maximum of five numeric codes are printed per source statement. Figure 3-2 shows a sample of this output.

3.2.3 Module Summary Output (Coding Standards Audit)

Following the audit of each module, Code Auditor prints messages showing the count of executable statements in the module in addition to the number of FORTRAN IF statements. Following is a list of FORTRAN statements included in the count of executable statements:

DO	PUNCH
GO TO	PRINT
IF	DECODE
PAUSE	ENCODE
STOP	READ
REWIND	WRITE
BACKSPACE	ASSIGN
ENDFILE	Assignment statements
CALL	Function calls
EXIT	
RETURN	

Code Auditor also outputs a table summarizing the results of the audit of the module. It is shown in figure 3-3. The numeric codes and descriptions on the left of the tables identify the standards not adhered to in the module. The "total errors" column contains counts of violations for each of the standards identified. The "card numbers" column


SOURCE CODE
CA NO.
SUBROUTINE ARRAY(IARGU) B002670 1
C B172680 2
CCCC

CODE AUDITOR SUMMARY FOR ROUTINE ARRAY

EXECUTABLE STATEMENT COUNT FOR THIS ROUTINE 11] - COUNT OF EXECUTABLE STATEMENTS IN MODULE

NUMBER OF IF STATEMENTS FOR THIS ROUTINE 2

ERROR NUMBER	DESCRIPTION	TOTAL ERRORS	CA-CARD NUMBERS
8	USE INTEGER SUBSCRIPTS WHEN REFERENCING AN ARRAY	1	18
9	DATA STMTS TO PRECEDE EXECUTABLE CODE	1	13
24	COMMON BLOCK LENGTH SAME IN ALL MODULES	2	10 11
26	NEVER OMIT INDEX EXCEPT CALL,DATA,OUTPUT STMTS	1	19
31	PREFACE COMMENT BLOCK NOT CONSISTENT WITH STANDARD	1	9
34	NO RETURN STMT CAN CONTAIN AN ARGUMENT LIST	1	29

STANDARDS NOT ADHERED TO — VIOLATED STANDARDS DESCRIPTION — TOTAL NUMBER OF VIOLATIONS — LIST OF CODE AUDITOR ASSIGNED CARD NUMBERS WHERE THE CORRESPONDING STANDARD WAS NOT ADHERED TO

Figure 3-3

lists the Code Auditor assigned card numbers where the corresponding standard has not been adhered to. A maximum of five (5) card numbers are listed for each violated standard.

3.2.4 Program Summary Output (Coding Standards Audit)

Following audit of all modules and following printout of the summary data for the last module audited, Code Auditor prints three tables containing summary data for module audited. Examples of the three tables appear in figures 3-4 through 3-6 following.

Subroutine "SUB1" has 3 instances
of non-adherence to standard no. 3

ROUTINE	RESTRICTION											
	1	2	3	4	5	6	7	8	9	10	11	. . .
CAREG	2	3	3	1	0	1	2	0	13	1	0	. . .
SUB1	0	2	3	0	0	0	1	0	3	0	0	. . .

Figure 3-4, Program Summary Output
(Error Summary)

The Code Auditor Error Summary Report (figure 3- 4) has one entry for each routine audited for each of the 37 standards.

ROUTINE	TOTAL ERRORS	TOTAL CARDS	PERFORMANCE INDEX
CAREG	60	153	60.78
SUB1	17	53	67.92
TOTALS	77	206	62.62

Figure 3-5, Program Summary Output
(Error Totals)

The Error Totals table (figure 3-5) has one line of data for each routine audited plus the line of data for the totals. The performance index is

$$\left(1 - \frac{\text{total errors}}{\text{total cards}}\right) \times 100, \text{ a grade from zero to 100.}$$

When the performance index is calculated to be less than zero Code Auditor outputs a value of zero.

STANDARDS VIOLATION TOTALS

RESTRICTION	1	2	3	4	5	6	7	8	9	10	11 . . .
TOTAL	0	0	6	0	0	0	0	0	0	0	0 . . .

There are 6 instances of non-adherence to coding standard number 3 in the routine audited.

Figure 3-6, Program Summary Output
(Standards Violation Totals)

The Standards Violations Totals table has one entry per standard 1 through 37.

3.2.5 Module Segmentation Output (Program Structural Analysis)

Following the auditing of the user's program for adherence to the RADC coding standards, the program's logical structure is examined. Appendix B describes Code Auditor's approach in checking the user's source code for conformance to program structural requirements. The segmentation data printed consists of a listing of the source code being audited, annotated in the left margin with Code Auditor assigned segment identification codes. The numeric codes assigned are sequentially ordered from the initial source statement of a

program module to the last statement. The numeric segment identifiers are not reinitialized if more than one module is processed; the initial segment number assigned for an intermediate module is one greater than the last assigned segment number for its predecessor module. Figure 3-7 following depicts a sample of this printout.

3.2.6 Segment Transfer Table Output (Program Structural Analysis)

The Segment Transfer Table, depicting the logical transfer of control among the program module's segments (3.2.5) is printed during this phase of output. Each Segment Transfer Table, comprised of ordered pairs of "from-to" segment identification codes, corresponds on a one-to-one basis with each program module processed. Tracing through the Segment Transfer Table should provide a string of segment identification codes for each logical path through the corresponding program module. In the Segment Transfer Table of figure 3-8, the two logical paths through the sample Segment Transfer Group are 31-32-34-35 and 31-33-35 respectively. This figure corresponds to the executable instructions and assigned segment identification codes for the sample segment identification group depicted in figure 3-7. An example interpretation of the Segment Transfer Table follows. If the decision portion (ISEG1.EQ.1) of the preceding IF expression is 'TRUE',

RADC CODE AUDITOR

PROGRAM SEGMENT DESCRIPTION

```
                    SUBROUTINE ARRAY(IARGU)                                  A002180
                    COMMON /IARY/IARY(10)                                    A042230
                    COMMON /WRGLN/WRGLN1                                     A112240
                    DATA J/3/                                                A052250
 31                 I=1                                                      A052260
                    I=2                                                      A042300
                    IARY(I)=1                                                A042310
                    ISEG1=1                                                  A182350
                    ISEG2=1                                                  A182360
  32                IF(ISEG1.EQ.1)GO TO 100                                  A182400
 33                 ISEG1=2                                                  A182410
                    GO TO 200                                                A182420
 34              100 ISEG2=2                                                 A182430
 35              200 CONTINUE                                                A182440
 36  37          300 IF(ISEG2.NE.2)GO TO 400                                 A182480
 38                 ISEG1=3                                                  A182490
                    GO TO 300                                                A182500
 39              400 CONTINUE                                                A182510
 40              500 ISEG1=4                                                 A182550
  41                IF(ISEG1.NE.4)GO TO 900                                  A182590
 42              600 ISEG1=5                                                 A182630
  43                IF(ISEG1.EQ.5)GO TO 700                                  A182670
 44                 ISEG2=5                                                  A182680
  45                IF(ISEG2.NE.5)GO TO 600                                  A182720
 46                 ISEG3=5                                                  A182730
                    GO TO 800                                                A182740
 47              700 ISEG4=5                                                 A182750
 48              800 CONTINUE                                                A182760
  49                IF(ISEG1.LT.1.AND.ISEG1.GT.4)ISEG1=4                     A182800
 50                 GO TO (900,1000,1100,1200),ISEG1                         A182810
 51              900 ISEG1=6                                                 A182820
                    GO TO 1300                                               A182830
 52             1000 ISEG2=6                                                 A182840
                    GO TO 1300                                               A182850
 53             1100 ISEG3=6                                                 A182860
                    GO TO 1300                                               A182870
 54             1200 ISEG4=6                                                 A182880
 55             1300 CONTINUE                                                A182890
                    RETURN                                                   A162930
                    END                                                      A002940
```

SAMPLE SEGMENT IDENTIFICATION GROUP

SEGMENT IDENTIFIER CODES

INPUT SOURCE CODE

Figure 3-7

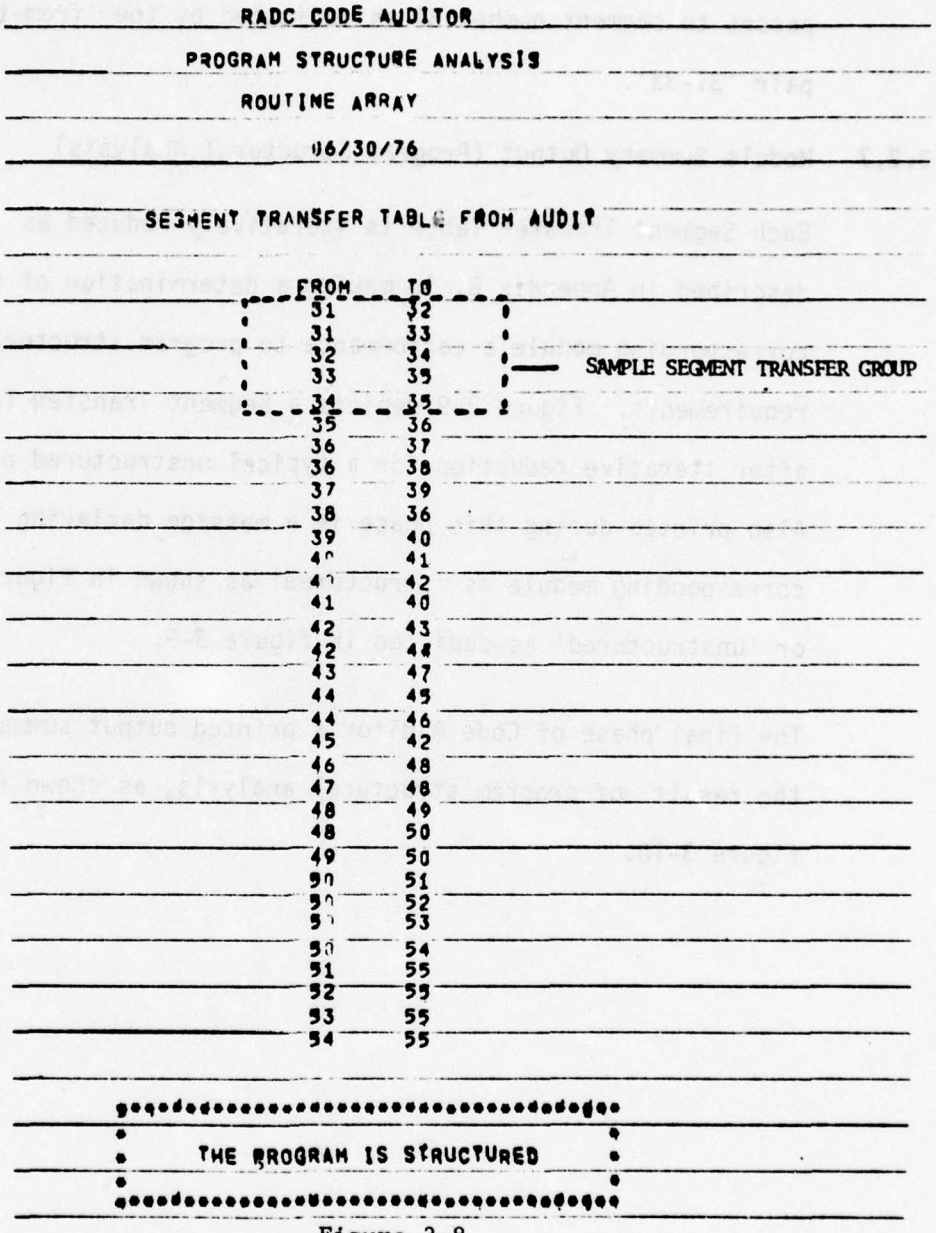

RADC CODE AUDITOR
PROGRAM STRUCTURE ANALYSIS
ROUTINE ARRAY
06/30/76
SEGMENT TRANSFER TABLE FROM AUDIT
FROM TO
31 32
31 33
32 34
33 35
34 35
35 36
36 37
36 38
37 39
38 36
39 40
41
42
41 40
42 43
42 44
43 47
44 45
44 46
45 42
46 48
47 48
48 49
48 50
49 50
51
52
53
54
51 55
52 55
53 55
54 55
SAMPLE SEGMENT TRANSFER GROUP
THE PROGRAM IS STRUCTURED

Figure 3-8

control is transferred to segment number 32 (GO TO 100) as indicated by the 'from-to' pair '31-32', otherwise control passes to segment number 33 as indicated by the 'from-to' pair '31-33'.

3.2.7 Module Summary Output (Program Structural Analysis)

Each Segment Transfer Table is iteratively reduced as described in Appendix B, in making a determination of the corresponding module's conformance to program structural requirements. Figure 3-9 depicts a Segment Transfer Table after iterative reduction for a typical unstructured program. Also printed during this phase is a message declaring the corresponding module as 'structured' as shown in figure 3-8, or 'unstructured' as depicted in figure 3-9.

The final phase of Code Auditor's printed output summarizes the result of program structural analysis, as shown in figure 3-10.

RADC CODE AUDITOR

PROGRAM STRUCTURE ANALYSIS

ROUTINE MAIN

06/30/76

SEGMENT TRANSFER TABLE FROM AUDIT

FROM	TO
1	2
2	5
3	4
4	5

THE PROGRAM IS NOT STRUCTURED

SEGMENT TRANSFER TABLE AT COMPLETION OF REDUCTION

FROM	TO
1	5
3	5

Figure 3-9

```
RADC CODE AUDITOR
PROGRAM STRUCTURE SUMMARY

06/30/76

ROUTINE          RESULTS
--------         ---------

MAIN             UNSTRUCTURED

ARRAY            STRUCTURED


2 ROUTINES WERE AUDITED
1 ROUTINES WERE STRUCTURED
1 ROUTINES WERE UNSTRUCTURED
```

Figure 3-10

3.2.8 Error Messages

In addition to its normal printed output, the Code Auditor prints a number of error messages. Many of the error messages printed are simply a result of sanity checks performed by the Code Auditor that suggests the user's source input is not compilable. Others are due to excessive module size causing Code Auditor's internal tables to overflow while the remaining messages are usually a result of logical errors not detectable at compile time. In any event, user response recommendations are provided for each error. Should the recommended corrective actions not clear the problem, the Code Auditor maintenance programmer should be consulted. A brief explanation of each error message plus recommended user response follow. Some error messages are grouped as a unit due to similarity in content and corrective action.

(1) Error Message - MORE THAN 100 TASKS

Description - The Code Auditor can process a maximum of 100 modules (main programs, subroutines, function subprograms, etc.) per execution. Excessive modules cause an abort.

Corrective Action - Break the source input into groups of 100 or less modules and process each group in separate executions.

(2) Error Messages (a) - XXX OVERFLOW - ABORT RUN

(b) CODE AUDITOR VARIABLE TABLE LENGTH EXCEEDED....SEE PROGRAMMER

(c) VSTR RATIO LESS THAN ONE

(d) ERROR - TOO MANY SEGBF ENTRIES

(e) TABLE OVERFLOW - THIS RUN CONTAINS XX TRANSFERS

Description - All of the above messages indicate the overflow of various tables internal to the Code Auditor and may all be attributed to excessive module size.

Corrective Action - Remove the offending module and consider its redesign and parceling. In all likelihood the module violates coding standard No. 6; maximum of 100 executable statements per module.

(3) Error Messages (a) - CLEANUP - FATAL ERROR - SEGMENT xx FROM SEGBT IS GREATER THAN ALL SEGTAB ENTRIES

(b) - CLEANUP - FATAL ERROR - LABEL xx NOT FOUND IN SEGTAB TABLE

(c) - INITSG ERROR - NAME xxx
NOT IN KTABLE

(d) - **ERROR** UNSOLVABLE PROBLEM
IN TRAPIT

Description - These errors are most likely logical errors not detected at compile time.

Corrective Action - Check logical program structure after recompiling.

(4) Error Messages (a) - ERROR DETECTED BY IFCK
IFCK ERROR 1
MATCHING PARENTHESES WERE NOT FOUND

(b) - *ERROR DETECTED BY SQZB*
SQZB ERROR 2
0 PRECEDES THE HOLLERITH STRING

Description - These errors detected as a result of sanity checks conducted by the Code Auditor, and usually indicate the source input is not compilable.

Corrective Action - Recompile prior to processing thru Code Auditor again.

3.3 Operating Procedures

The Code Auditor was designed to execute in a Honeywell 600/6000 batch environment under GCOS control. Three control card decks are described for executing the Code Auditor under GCOS. Each deck corresponds to one of the three alternative input mediums on which the user's source code may reside. The following common rules apply to all control cards:

(1) All control cards except the ***EOF card are identified by a $ in card column 1.

(2) The control card name (i.e., IDENT, LIMITS, TAPE, etc.) begin in card column 8.

(3) The variable field begins in card column 16 and must not exceed column 72.

(4) Variables must be separated by a comma.

(5) A blank terminates the field definition and the card scan. Therefore, no embedded blanks are permitted.

(6) Upper case alphabetic entries are required; lower case are user/installation supplied data.

3.3.1 Source Input via Cards

The control card sequence below shows the deck structure necessary to execute with source input from cards.

Control Card	Description
$ IDENT account no., identification	Identifies the user of a job and supplies the user's account no. The variable field format and content is installation dependent.
$ USERID system master catalog name	Identifies the Code Auditor system's master catalog name. The variable field's format and content is installation dependent.
$ EXECUTE DUMP	Requests the loading and subsequent execution of the Code Auditor program. Should the execution activity terminate abnormally, the DUMP parameter in the variable field request a slave core dump.

Control Card	Description
$ LIMITS 05, 40K, 9K, 5K	Modifies the default resource limits. Interpretation of the variable field parameters follow: 05 the maximum processor run time is 5 minutes. 40K the maximum core storage for running the Code Auditor is 40K words. 9K the amount of core storage that can be shared with the General Loader when the Code Auditor is loaded is 9K words. 5K the maximum number of lines to be written on SYSOUT during program execution is 5K lines.
$ PRMFL H*, permit, mode, file string	Accesses a permanent file where the relocatable object code for the Code Auditor resided. Variable field entries are installation supplied.

Control Card	Description
$ FILE 11, A3RR, 1L	Sets up allocation of a mass storage file. The logical unit designator is 11.
$ DATA 05	Indicates that program input data immediately follows this card. The Code Auditor will reference this file with a logical unit designator of 5.
CAOPTION parameters	This is the option card input to the Code Auditor. See 3.1.2 for format and content.
$ DATA 13	Indicates that program input data immediately follows this card. The Code Auditor will reference this file with a logical unit designator of 13.
User's Source Code	The user's source should follow immediately the card, $ DATA 13.
$ SYSOUT 06	Assigns the Code Auditor's printed output (logical unit number 6) to SYSOUT for on-line conversion.
$ ENDJOB	Indicates end of job and must be the last '$' control card of the job.

3.3.2 Source Input From Disk

The control card sequence below shows the deck structure necessary to execute with source input from disk.

Control Card	Description
$ IDENT account no., identification	See 3.3.1
$ USERID system master catalog name	" "
$ EXECUTE DUMP	" "
$ LIMITS 05, 40K, 9K, 5K	" "
$ PRMFL H*, permit, mode, file string	" "
$ FILE 11, A3RR, 1L	" "
$ DATA 05	" "
CAOPTION parameters	See 3.1.2
$ FILE 13, device name, access	Identifies the disk file containing the user's source code. Except for the file code of 13, all other parameters are user supplied. See 1.2 (1).
$ SYSOUT 06	See 3.3.1
$ ENDJOB	" "

3.3.3 Source Input From Tape

The control card sequence below shows the deck structure necessary to execute with source input from tape.

Control Card	Description
$ IDENT account no., identification	See 3.3.1
$ USERID system master catalog name	" "
$ EXECUTE DUMP	" "
$ LIMITS 05, 40K, 9K, 5K	" "
$ PRMFL H*, permit, mode, filing string	" "
$ FILE 11, A3RR, 1L	" "
$ DATA 05	" "
CAOPTION parameters	See 3.1.2
$ TAPE 13, device name, multireel indicator	See 3.3.1 Assigns a tape unit to the users input source code. Except for the file code of 13, all other parameters are user supplied. See 1.2 (1).
$ SYSOUT 06	See 3.3.1
$ ENDJOB	" "

APPENDIX A

RADC FORTRAN CODE AUDITOR CODING STANDARDS

A brief description of the standards incorporated in the current version of the RADC FORTRAN Code Auditor Program follows:

1	Not used
2	Stmt labels are to be in columns 2-5, right justified
3	Stmt labels are to be in ascending order
4	Continuation stmts to be sequenced: 1-9, A-J
5	Not used
6	Maximum of 100 executable statements per routine
7	Not used
8	Use integer subscripts when referencing an array
9	DATA stmts to precede executable code
10	One type specification for a variable name
11	Arguments in CALL stmts must not contain arith, or logical expressions
12	Real time routines may not have arguments in CALL stmts
13	One assignment statement per line
14	No mixed mode arithmetic on right side of equal sign
15	Whole numbers used as exponents must be integer
16	Not used
17	Not used
18	DO loop nests must not exceed six levels
19	FORMATS are placed after I/O statements, or after RETURN
20	Not utilized
21	Not utilized
22	I/O devices must be referred to by integer variable name
23	Computed GOTO stmts allowed if GOTO variable is checked
24	COMMON block length must be same in all modules
25	All COMMON blocks will be labeled
26	Never omit array subscripts except in DATA, CALL or output statements
27	STOP and PAUSE stmts not allowed in real-time programs
28	Assigned GOTO stmts are not allowed in real-time programs
29	Not used
30	Not used
31	Preface commentary block to contain:

C or * in column 1 on all cards

First card:	* in cols 2 or 3 thru 71 or 72
Intermediate cards:	Same symbol in col 1 as first card followed by text
Last card:	* in same columns as first card

32A Comments must precede blocks of one or more IF stmts
32B Comments must precede blocks of one or more CALL stmts
32C Comments must precede blocks of one or more I/O stmts
32D Comments must precede mixed mode assignment stmts
33 In-line comments consist of three or more cards:

C or * in column 1 on all cards	
First card:	Blanks in cols 2-72
Intermediate cards:	Text
Last card:	Blanks in cols 2-72

34 No RETURN stmt can contain an argument list
35 Not used
36 Not used
37 Not used

The following are the reasons for the standards enforced by the Code Auditor. Numbers correspond to the list of standards.

2) This restriction is enforced to ensure better readability in source code and forces a consistency in all programs.

3) Placing statement labels in ascending order will force the code to be more readable by using labels to sequentially illustrate the routine/subroutine. Developers are also encouraged to use statement labels for illustrating specific "blocks" or "loops" in code, (e.g., use order of magnitude of statement label numbers to illustrate "loops" or "blocks").

4) This restriction is enforced to ensure better readability in source code and forces a consistency in all programs.

6) This restriction enforces the philosophy of modular programming.

8) This restriction prevents mixed mode processing of array subscripts. This also prevents octal or fixed point indexing which is prone to error.

9) DATA specification statements are placed here to secure a uniform location in routine/subroutine code where data specifications may be found. Placing DATA specification statements here also ensures that they will not be placed after executable code.

10) This restriction facilitates readability. This practice also prevents redefinition of specifications from conflicting.

13) This restriction enhances readability and eliminates confusion.

14) Mixed mode is inefficient because it forces a penalty in execution time to make mixed mode conversions.

15) Integer exponentiation is used throughout to preserve accuracy in potential conversion roundoff errors.

18) This results in inefficient code because the compiler will not optimize code nested this deep.

19) FORMAT statements are placed at this position to define a uniform location in all code where FORMATS referenced from I/O statements may be found.

22) This restriction is enforced to prevent mode processing of I/O statements. Reference by variable name allows easy change in program.

23) This restriction forces 'computed GO TO' usages to be completely bounded logically and aid in ease of reading.

24) This restriction ensures proper correspondence in COMMON block.

25) All COMMON blocks in code are labeled to avoid the risk and confusion of misinterpretation of data base locations.

26) This is enforced to keep flexibility in the processing of array subscript operations.

27) STOP/PAUSE statements are not allowed in real-time programming.

28) The philosophy behind structured programming is that it encourages more efficient code due to less branching and jumping to different places in code. This restriction is intended to enforce this philosophy.

31) This standard is enforced to ensure a consistency in preface commentation of all programs.

32) Comments must precede "IF" tests, branch to statements, input/output statements and statements containing statement labels to flag and define significant portions of the source code. This practice will also force better documentation.

33) This restriction is enforced to keep commentation in the body of routine/subroutine code consistent in all programs.

34) RETURN statements do not contain argument lists. Orderly subprogram exits are enforced.

APPENDIX B

STRUCTURED PROGRAM REQUIREMENTS

This appendix defines what is meant by the term "structured program" and describes the analytical methods employed by the Code Auditor in examing a user's source code for adherence to "structured program" requirements.

B.1 Program Segmentation

The initial step taken in determining whether a program module adheres to "structured program" requirements involves the syntactical analysis of each of the module's source statements.

This examination results in an identification of the logical structure of the module and its constituent parts (segments). A segment, for our purposes, is defined as a sequence of statements with one 'entry' statement and one 'exit' statement (which may or may not be the same statement), such that if the 'entry' statement in the sequence is executed, all statements in the set are executed; the 'exit' statement being executed prior to the transfer of execution control to another segment. Thus, in the sequence of source statements which follow, the group of statements between the expression A=B, and the 'predicate' portion of the IF expression, inclusive, would comprise a single segment (assigned a segment

identification code of 1).

```
 5   A=B
     B=B+1.2
     IF (A .EQ. C) GO TO 20
     C = C - 1.5
     .
     .
     .
     .
     B = A+B+C
20   A = A+1.0
     .
     .
     .
```

Segment transfer may then be defined as the passing of execution control, either implicitly or explicitly, from one segment to another. In the preceding example, the statement IF (A.EQ.C), implicitly passes execution control to the statement GO TO 20 (assigned a segment identification code of 2), if the 'IF' condition is 'TRUE', otherwise implicit transfer is made to the statement C=C-1.5 (segment identification code is 3). The 'GO TO' statement explicitly passes execution control to the statement labelled '20' (assigned a segment identification code of 4). It should be noted that the statement GO TO 20, is a one statement segment, i.e., comprises both the 'entry' and 'exit' statements of segment no. 2.

In order to automatically identify a program module's segments, a segment's entry statement must be defined in a completely unambiguous manner. A source statement is defined as a segment 'entry' statement if:

(1) it is the first executable statement of a program module.

(2) it is the first executable statement following a FORTRAN ENTRY statement.

(3) it is the first executable statement following a subroutine CALL statement.

(4) it is the first executable statement following a logical or arithmetic IF statement.

(5) it contains a FORTRAN label (FORMAT statements excepted) whether referenced or not.

(6) it is a DO statement.

(7) it is the first executable statement following a DO terminator.

(8) it is the 'consequent' statement of a logical IF statement.

Similarly a source statement is defined as a segment 'exit' statement if:

(1) it is an unconditional GO TO statement.

(2) it is a computed GO TO statement.

(3) it is an assigned GO TO statement.

(4) it is an arithmetic IF statement.

(5) it is the 'predicate' portion of a logical IF statement.

(6) it is a CALL to an external routine.

(7) it is a RETURN statement.

(8) it is a STOP statement.

(9) it is the terminal statement of a DO loop.

(10) it is any other executable which precedes an segment 'entry' statement.

B.2 Segment Transfer Table

Upon completion of program module syntactical analysis and identification of the module's logical segments, the Code Auditor builds a table which describes the interrelationship of these segments. There is one table per program module processed. Describing the interrelationship among a program module's segments simply means the 'from-to' pairing of segment identification codes involved in the transfers of execution control, thus the table name 'Segment Transfer Table'. The Segment Transfer Table shown below depicts the segment interrelationships of the example program statements of Section B.1.

FROM	TO
1	2
1	3
2	4
3	4

The reader may verify the authenticity of the over-simplified example.

B.3 Structural Analysis

To begin our definition of a "structured program" consider the coding structures shown in Figure B-1. A "structured program" may contain a combination of only those code structures depicted in Figure B-1, such that control flows from top to bottom or from beginning to end. Back-tracking is not allowed. To check if a program module complies with "structured program" requirements, the Code Auditor iteratively applies a reduction algorithm to the module's corresponding Segment Transfer Table until the residual table is irreducible. If the residual table contains more than one entry, then the module is unstructured. The algorithm steps are as follows:

(1) Delete all trivial entries; entries of the form (i,i)

(2) Delete all redundant entries; entries of the form (i,j) except the first

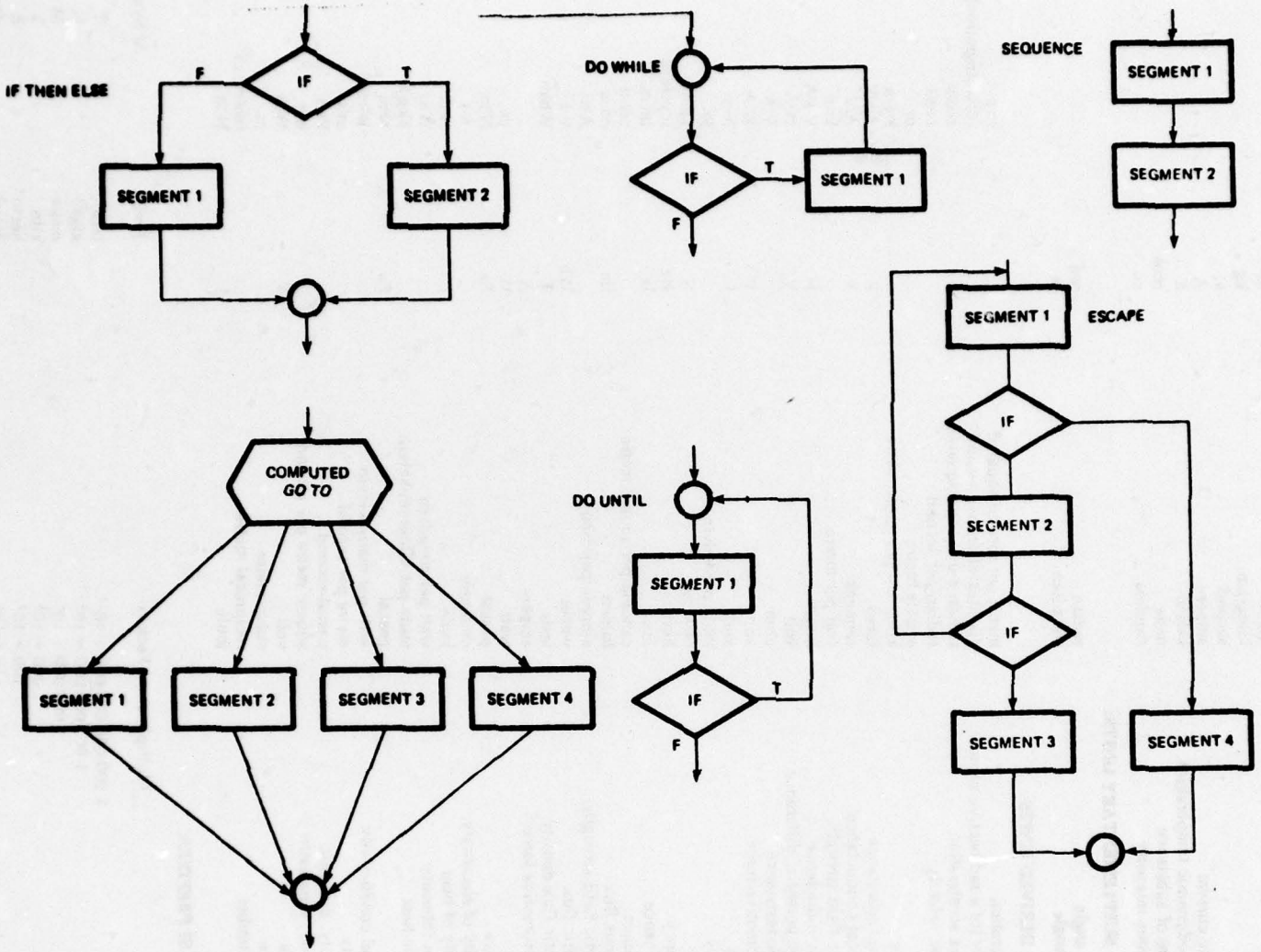
(3) If i transfers to j and either

  (a) j only transfers to k for some k

  or

  (b) i only transfers to j

  Then: replace j with i



STRUCTURAL ANALYSIS CODING STRUCTURES

FIGURE B-1

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

*U.S. GOVERNMENT PRINTING OFFICE: 1977-714-088/9