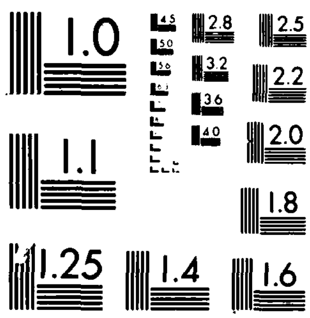
AD-A124 030 GUIDELINES FOR CODING FORTRAN PROGRAMS(U) NAVAL OCEAN 1/1
RESEARCH AND DEVELOPMENT ACTIVITY NSTL STATION MS
J J CORNYN JUL 82 NORDA-41

UNCLASSIFIED F/G 9/2 NL

END
DATE
FILMED
3-84
DTIC



(12)

NORDA Report 41

AD A 124030

Guidelines for Coding FORTRAN Programs

John J. Cornyn

Numerical Modeling Division
Ocean Science and Technology Laboratory

July 1982

DEPARTMENT OF THE NAVY
NAVAL OCEAN R&D ACTIVITY

Approved for Public Release
Distribution Unlimited

Naval Ocean Research and Development Activity
NSTL Station, Mississippi 39529

Foreword

This guideline is designed to assist individuals in writing FORTRAN programs. Adherence to the conventions described herein should lead to readable and maintainable programs. In addition, it should significantly reduce the amount of time and effort required to transfer a program from one computer to another. Although this document was written to serve as a coding guideline, which the Acoustic Modeling Manager of the Surveillance Environmental Acoustic Support (SEAS) Project could provide to contractors and other Navy organizations supporting the SEAS effort, it could also be used, without modification, by any organization writing, or contracting for, FORTRAN programs.

G. T. Phelps

G.T. PHELPS, Captain, USN
Commanding Officer, NORDA

AD-A124 030

Guidelines for Coding Fortran Programs

Naval Ocean Research
NSTL Station, MS

July 82

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NORDA Report 41
2. GOVT ACCESSION NO: AD/A124030
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): Guidelines for Coding FORTRAN Programs
5. TYPE OF REPORT & PERIOD COVERED: Final
6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): John J. Cornyn
8. CONTRACT OR GRANT NUMBER(s):
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Ocean Research and Development Activity, NSTL Station, Mississippi 39529
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: PE63759N
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Ocean Research and Development Activity, Code 320, NSTL Station, Mississippi 39529
12. REPORT DATE: July 1982
13. NUMBER OF PAGES: 57
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

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)

18. SUPPLEMENTARY NOTES

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

FORTRAN coding
FORTRAN guidelines
computer programming standards
software configuration management
software maintenance

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

This document provides a set of conventions to be followed when writing FORTRAN programs. Enforcement of and adherence to these conventions should minimize problems in transferring programs to other computers, should lead to more readable programs, and should make programs more maintainable. For ease of reference, the guidelines have been arranged in the order the subjects they apply to would normally be covered in a FORTRAN reference manual. The final sections contain a FORTRAN program before and after the guidelines were

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) applied, and describe techniques for implementing structured programming constructs.

Executive Summary

This document provides a set of conventions to be followed when writing FORTRAN programs. Enforcement of, and adherence to, these conventions should reduce problems in transferring programs to other computers, should lead to more readable programs, and should make programs more maintainable. For ease of reference, the guidelines have been arranged in the order that the subjects to which they apply would normally be covered in a FORTRAN reference manual. The final section shows the text of a FORTRAN program both before and after applying the guidelines. An appendix describes how to emulate with FORTRAN code the constructs of structured programming.

Acknowledgements

The author is indebted to numerous individuals. Many of the ideas presented appear in works by J. E. Fleiss, G. W. Phillips, A. Edwards, L. Reider (Fleiss, 1974), H. F. Ledgard and L. J. Chmura (Ledgard, 1978), and G. Jacobs (Jacobs, 1976). The author especially wishes to thank Richard Lauer of NORDA and Gil Jacobs of Ocean Data Systems, Inc. (ODSI), Rockville, Maryland, for reviewing the document, engaging in discussions regarding its content, and offering numerous helpful suggestions. Finally, the author is indebted to CDR Kirk Evans, formerly Acoustic Modeling Program Manager, Surveillance Environmental Acoustic Support (SEAS) Project, NORDA Code 522, for funding this work, Program Element 63759N.

Contents

Contents

Contents

Contents (Continued)

Guidelines for Coding FORTRAN Programs

1. Introduction

1.1 Scope

This document provides a set of conventions to be followed when writing FORTRAN programs. Adherence to these guidelines should make it easier for programmers to understand the programs they write, especially when they review them three weeks, three months, or three years after their inception. This increased understanding will also allow developers to concentrate on solving the problems that their programs were originally designed to address, rather than resolving the problems created by programming in an arbitrary, undisciplined style. Adherence to these guidelines should make program maintenance less time-consuming for the individual, and less expensive for the organization chartered with this responsibility. Adherence to these guidelines should also facilitate transferring a program from one computer to another.

Most of the conventions contained here are concrete and specific to facilitate enforcement.

This document assumes that the reader is conversant in FORTRAN, and thus does not attempt to explain the purpose or the meaning of FORTRAN statements and associated constructs. Readers interested in achieving a better understanding of the syntax and semantics of the FORTRAN statements mentioned herein are referred to the books by McCracken (1972a) and Control Data Corporation (1976) mentioned in the reference section of this report.

The American Standards Institute (ANSI) USA Standard FORTRAN X3.9-1966 was used as a starting point in writing this document. Photocopies of this standard may be purchased from the American National Standards Institute, Inc., 1430 Broadway, New York, NY 10018 (Telephone: (212) 354-3300). At the time of writing, X3.9-1966 cost \$24.95, and X3.9-1978 cost \$16.50, with a \$4.00 shipping charge. USA Standard Basic FORTRAN X3.10-1966 was not used because most programs for the underwater acoustics community do not run on the small systems for which this standard was designed. Although a more recent FORTRAN standard (ANSI Standard X3.9-1978, sometimes referred to as FORTRAN 77) exists, we have chosen not to use it because many FORTRAN compilers in the Navy do not support it, even though it was declared the approved standard on 3 Apr 78 and X3.9-1966 was withdrawn. The new FORTRAN Standard was designed, however, to minimize conflicts with X3.9-1966. We have modified this report to further reduce these conflicts. Unfortunately, adherence to these standards does not guarantee that programs will be written clearly or concisely, or will have a well-structured design. And X3.9-1966 does not permit many desirable FORTRAN constructs, such as specification of Hollerith characters without character counts. To achieve maximum transferability of software, developers must consider other factors not addressed by the standard, and even avoid the use of some statements permitted by the standard. Because many programmers may not be exactly sure what is, or what is not, permitted by the ANSI standard, this document comments on many commonly used constructs which are not permitted by the standard, and specifically states when they must be avoided.

This document was developed by considering coding conventions suggested in the works of Berkowitz (1976), Fleiss

(1974), Jacobs (1976), Ledgard (1978), McCracken (1972a and 1972b), Roberts (1969), Jensen (1979) and Yourdon (1975), as well as the author's programming experience. At times, it was necessary to choose between conventions that appeared to have equal merits, but which were either conflicting or inconsistent with one another.

For ease of reference, the guidelines have been arranged in the order that the subjects would normally be covered in a FORTRAN reference manual. Preceding each coding convention described in this document is a letter: M, S, or L. An M next to a convention indicates that it must, in the author's opinion, always be followed. Any violations of such a convention must be approved by the individual in the organization who is responsible for enforcing the coding guidelines. An S indicates the coding convention should be followed whenever possible. An L indicates that it would be helpful to follow this convention because it most likely would improve program clarity.

Section 11 provides a summary of the possible FORTRAN statements, indicates the sections of this document where they are discussed, and offers overall recommendations concerning their use.

Section 12 is divided into two parts. The first part shows the original coding of an ambient noise model (CNOISE). This code was written prior to the formulation of the guidelines suggested herein and is an actual model, rather than an example designed exclusively for this report. It is typical and, in many respects, better than most of the coding found today. Its major deficiency is the general lack of comments. The second part shows this same program after being revised to abide by the guidelines described herein.

Appendix D describes in detail how to emulate the control structures of structured programming (Jensen (1979), Yourdon (1975)). Exclusive use of these structures is highly recommended. Their value has been well-established by the software engineering community.

This document was designed to serve as a guideline to be followed by private contractors and other organizations when writing FORTRAN programs in support of the SEAS Project Office. The need for this document was clearly indicated by the lack of attention many software developers gave to the clarity, transportability, and maintainability of their progams. This neglect needlessly resulted in high program conversion costs, numerous maintenance headaches, and untold hours of wasted time trying to decipher the meaning of uncommented, spaghetti-like logic. Since other organizations have similar problems, it is hoped that this document will prove to be useful to them as well. It is believed that even a casual reading of this document will improve most programmers' style.

2. Coding Fortran Statements

2.1 General Remarks and Suggestions

M - All source code must (whenever possible) be written in a subset of American National Standards Institute (ANSI) FORTRAN, as described in X3.9-1966. The limits of this subset are defined below.

M - Extensions to the above standard must not be used, unless implementation of the program is impossible without their use. (Some manufacturers, e.g., Control Data Corporation, have compilers that will check for and flag noncompliance with the ANSI standard.)

M - If the conventions described herein are adopted as standards by an organization, any violation of those conventions designated by the letter M must be approved by the individual appointed to enforce them. Standards that are not enforced are useless.

M - Don't subvert the intended purpose of the language, i.e., avoid programming tricks.

M - Do not write programs that modify themselves as they execute.

S - For each installation and application, users should follow an adopted set of standard variable names. This procedure will make programs easier to understand and interface, if required, at a later time.

S - Build as many error checks into the program as possible, under the assumption that it will do something incorrectly. This will help reduce debugging time.

S - Avoid having one module (group of statements that perform a specific function) "fall" into another. It is better to have explicit transfers between modules than to rely upon the sequential execution of code.

S - Adopt the KISS (Keep It Simple, Stupid) philosophy. Always use the simplest language features that will solve the problem adequately.

S - Design the program listing so that it is pleasing to the eye.

S - Avoid including more significant digits in the output of a program than can be justified by its input.

2.2 FORTRAN Character Set

M - Do not use characters other than:

= equal sign	) right parenthesis
+ plus sign	, comma
* asterisk	. decimal point
/ slash	- minus sign
(left parenthesis	blank
alphabetic (A-Z)	numeric (0 to 9)
' apostrophe	

Only these characters are permitted by the ANSI standard X3.9-1978. The quote (") and not-equal (≠) symbols are not permitted. The currency symbol ($) was permitted by X3.9-1966 but the apostrophe (') was not.

2.3 FORTRAN Statements

M - FORTRAN keywords must be clearly set off by a blank character or other separator to improve program clarity.

M - The executable statements in the range of a DO loop must be indented at least three spaces from the DO statement and terminating CONTINUE statement.

S - FORTRAN statements should be numbered in columns 73-80. This facilitates communication of updates and reconstruction of dropped decks.

S - All elements related to a specific level of the control structure should be aligned to the same column.

S - All dependent processes associated with a structure should be indented by a fixed amount, usually two to five spaces. The optimum amount is three spaces. The dependent process could also be a control structure, in which case processes subordinate to the dependent control structure should again be indented to indicate their relative position in the structure.

2.4 Continuation Lines

M - Do not use more than nine (9) successive continuation lines. Nineteen is the maximum permitted by the ANSI standards 3.9-1966 and X3.9-1978. But the subset language of X3.9-1978 permits only nine.

M - Continuation lines must be designated by nonzero alphanumeric characters in column 6. They must be numbered sequentially, starting with 1 through 9, then, if needed, go A through K. Column 7 of a continuation line must always be left blank, except when this prevents outputting a Hollerith string begun on the previous line. This step is especially important in

FORMAT statements, since the integers in column 6 could be adjacent to integers in format fields, thereby reducing readability. For example, 25X where the 2 is in column 6 might be incorrectly interpreted as "skip 25 blanks."

M - Columns 1-5 of a continuation line must contain blanks. This step is included to increase agreement with X3.9-1978.

2.5 Statement Separator

M - Do not put more than one statement on a line. Use of more than one statement per line can make programs extremely difficult to read. Also, it is not permitted by the ANSI standard.

2.6 Statement Labels

M - Statement labels (numbers) on statements (other than FORMAT statements) which are not transferred to from other points in the program must be eliminated from the source code. Extraneous labels make programs difficult to read and debug.

M - Assign statement numbers in ascending sequence of value, initially by 10's or 100's, so that additional statements can be inserted without renumbering.

M - Statement-label numbers must appear only on CONTINUE and FORMAT statements; e.g.,

```
      100 CONTINUE
          B = SQRT(A+C)
not
      100 B = SQRT(A+C)
```

M - Do not put a statement number on an END statement. Its use on an END statement is not permitted by ANSI standard FORTRAN.

M - The statement labels used in a control statement must be associated with executable statements within the same program unit in which the control statement appears.

S - Keep statement numbers from one to four digits in length. Although ANSI X3.9-1966 FORTRAN permits up to five digits, four digits seems adequate and helps to prevent errors due to digits accidentally being typed in column 6.

L - Make all FORMAT statement numbers begin with the digit 9 and total four digits.

L - Statement-label numbers should be left justified in columns 2 through 5. This will help to prevent errors that can occur when one of the digits is accidentally punched in column 6. Also by starting in column 2, the first digit stands out more clearly than if the number started in column 1, since often there are comment cards, with Cs in column 1 before and after this card.

2.7 Comments (Also see Sect. 8.1)

M - All comment cards must begin with C in column 1. Other comment indicators, such as /, *, and $, are not permitted by the ANSI standard.

M - Programs must be divided into sections, each of which performs a given task. Before each section, comment cards must briefly describe the task being done by the section.

M - Include enough explanatory comments so that the reader can easily follow the code. Consider comments to be the equivalent of the text of a book. A well-documented FORTRAN program will often have more comment statements (C-statements) than executable statements. Contrary to some people's belief, comment statements do not slow down the execution of a program. Their absence, however, significantly impedes human comprehension.

M - Groups of comment statements must be set off by at least one blank comment card before and after the group.

M - Write comment statements so that they will help the reader to understand the program. A program comment of the form "add 1 to N" before the statement N=N+1 is no help. The goal is to anticipate the questions the reader will have and to answer them in advance.

M - Display comments so that they will stand out clearly.

M - Do not refer to specific statement labels (numbers) in comment statements or in FORMAT statements (for debugging printout for example). Because the program may be resequenced later, such comments and printed messages could, as a consequence, become incorrect and grossly misleading.

M - Each major decision point in a subroutine or main program must have comment cards explaining the decision.

M - Do not extend the text of comments into columns 73 through 80, because these columns may be used to sequence the card deck at a later time.

S - Arrange code and comments into visual groups reflective of the program logic.

S - Break up long routines into subdivisions. Each subdivision might correspond to a chapter in a mathematical textbook, with numerical labels and subheadings. Display these subdivisions by using special columns and "ruling lines across the page," that is, comment statements containing complete rows of dashes, asterisks, etc. Actually, if routines are well written they should not be so long as to make this procedure necessary.

S - Whenever a difficulty can be foreseen, insert a "warning" comment which is easily visible in the listing, e.g., by using asterisks or by punching WARNING in columns that are normally blank.

S - When using identation, keep the spacing conventions as consistent as possible.

S - Programs should be "commented" as they are written, not afterward. This will help expose errors in logic and inadequacies in the code. Also it should be easier to annotate a piece of code while it is fresh in one's mind as opposed to several days (or weeks) after it is written.

S - Meaningful comments should be added to critical sections of code, but care should be taken not to detract the reader's eye from the code itself.

L - Use blank space liberally, both inter-line and intra-line.

2.8 Blank Lines

M - Do not use blank lines. Instead use a blank line with a C in column one. Some compilers do not accept completely blank lines.

M - Make heavy use of blank lines (with a C in column one) to improve program clarity. They are especially useful for separating modules of a program and highlighting critical sections.

3. Language Elements

3.1 Constants

M - Use only integer, real, double precision, complex, logical, and Hollerith constants. Only these constants are permitted by ANSI standard X3.9-1966.

S - Whenever possible, constant data items should be isolated in DATA statements. This separation leads to greater readability and easier program modification.

3.1.1 *Integer Constants*

No recommended guidelines.

3.1.2 *Real Constants*

M - Do not divide by 0.0, as it is not permitted by the ANSI standards. Division by zero may result in values such as 0., 10^{-41}, etc., depending on the system.

3.1.3 *Double Precision Constants*

No recommended guidelines.

3.1.4 *Complex Constants*

No recommended guidelines.

3.1.5 *Octal Constants*

M - Do not use octal or hexadecimal constants, such as 525B, in FORTRAN programs. These, unfortunately, are not permitted by the ANSI standard. This constraint is indeed regrettable because an octal constant can often be more easily understood than a real constant when doing operations such as masking and shifting. Their use, however, could reduce the transferability of the program.

3.1.6 *Hollerith Constants*

M - Hollerith constants used in expressions and data statements must not have more than four, nor less than one, characters, e.g., 4HABCD is permissible but 6HABCDEF is not. This restriction is due to the limited word length of some machines.

M - Do not use right-justified with binary zero fill (NRf) or left-justified with binary zero fill (NLf) Hollerith constants, e.g., do not use

A = 4LABCD or A = 4RABCD.

Although these constructs are very valuable, they are not allowed by ANSI standard X3.9-1966.

M - Do not use Hollerith constants in statements other than in the argument of a CALL statement or in a data statement. Their use in other statements is not permitted by the ANSI standard X3.9-1966.

S - Avoid use of the apostrophe or other characters, such as * or $ to establish limits of size of Hollerith constants, e.g., A = 'ABCD'. Although this construct is extremely valuable and time-saving, it, regrettably, is not permitted by ANSI Standard X3.9-1966. Apostrophes are permitted by ANSI X3.9-1978 (see section 9.4).

3.1.7 *Logical Constants*

M - Do not use .T. or .F. to designate logical constants, use only .TRUE. or .FALSE. The abbreviated forms are not permitted by ANSI Standard X3.9-1966.

M - Put a blank character on either side of a logical constant to improve readability. For example,

```
LOGICAL  X1, X2
. . .
. . .
   X1    =  .TRUE.
   X2    =  .FALSE.
```

3.2 Variables

M - Variable names must have no more than six alphanumeric characters and no special characters. This restriction is unfortunate, but necessary, due to the inadequacies of some FORTRAN compilers and the definition of ANSI standard FORTRAN X3.9-1966.

M - The characters of a name must not be separated by blanks.

M - The first character of a variable name must be alphabetic.

M - The standard naming conventions, namely, using the letters A through H and O through Z as the first characters of real variables, and I through N as the first characters of integer variables, must be used and not overridden.

M - Variables that are never used in a program or a routine must be eliminated. Their presence reduces program readability and wastes storage space.

M - Variable names must not correspond to FORTRAN keywords. A table of FORTRAN keywords is provided in Appendix A.

M - Variable names must be as descriptive as possible to improve readability, e.g., the variable name for the number of ships should be NSHIPS, not simply N, or NS.

M - All variables must be clearly defined before being used in statements. Programmers must not assume that variables are initialized to zero (or any other value) at the beginning of a job.

M - Do not put part of a variable name on one line and the rest of it on a continuation line below. Keep variable names on the same line.

S - Variables ending with the letter O should be avoided. This letter has a tendency to be confused with the number zero.

S - One should not use the same variable name in different contexts to save memory space. Doing so is confusing to the reader and leads to massive confusion if someone later changes any of the meanings. Modern computers typically are never so storage-space-limited that this practice is justified, except possibly for very large arrays.

S - Use symbolic variables, rather than integer constants, for array dimensions in the text, e.g., in DO-loops, IF statements, I/O lists, etc. Put them all together in COMMON, and initialize them all in one routine. Then only this routine and the COMMON/DIMENSION statements need be altered if the dimensions need to be changed, e.g., to put the program on a smaller machine.

S - Avoid changing variable names across subroutine boundaries. For example, when passing a parameter from one routine to another, do not change its name unless it is unavoidable. Changing names across boundaries makes the program less readable.

3.2.1 *Integer Variables*

M - All integer variables must begin with the letter I, J, K, L, M, or N.

S - In general, counters (variables which are incremented and tested) should be integer. The incrementing and testing of integers is faster than the incrementing and testing of floating point (real) numbers.

3.2.2 *Real Variables*

M - All real variables must begin with one of the letters A through H, or O through Z.

3.2.3 *Double Precision Variables*

M - Do not use the implicit declaration of double-precision variables.

3.2.4 *Complex Variables*

M - Do not use integer complex variables. They are not permitted by the ANSI standard.

3.2.5 *Logical Variables*

No recommended guidelines.

3.3 Arrays

S - The total storage for any one array should not exceed 32767 (decimal) words.

S - The number of subscripts of an array in a calling program should be the same as the number of subscripts of the corresponding array in the called subroutine. For example, passing a two-dimensional array to a one-dimensional array via a subroutine call is a confusing and error prone practice and should be avoided. That is, avoid constructs such as

```
PROGRAM       A
DIMENSION     B(10,10)
. . . .
  CALL        SUB1 (B)
. . . .
END
SUBROUTINE    SUB1 (C)
DIMENSION     C(100)
. . . .
. . . .
RETURN
END
```

This is not a "must" because there are instances, where this procedure is very valuable, such as when one wishes to operate on columns of a multidimensional array with a column-oriented subroutine. Also the indexing problem is minimal, since virtually every machine stores arrays in the same column-wise manner in accordance with the following table from ANSI Standard X3.9-1966:

Value of a Subscript

Dimensionality	Subscript Declarator	Subscript	Subscript Value	Maximum Subscript Value
1	(A)	(a)	a	A
2	(A, B)	(a, b)	a + A * (b - 1)	A*B
3	(A, B, C)	(a, b, c)	a + A * (b - 1) + A * B * (c - 1)	

NOTES
(1) a, b, and c are subscript expressions.
(2) A, B, and C are dimensions.

3.3.1 *Subscripts*

M - The number of subscripts in an array appearing in an executable statement must always equal the number of declared dimensions of the array, except in an EQUIVALENCE statement.

M - Do not use expressions other than the following for subscripts:

```
C*V+K
C*V-K
C*V
V+K
V-K
V
K
```

where V is an integer variable, and C and K are integer constants. These are the only subscript expressions permitted by ANSI standard X3.9-1966.

M - Subscript bounds must never be exceeded. A subscript must never be given a value less than one or larger than the maximum length specified by the upper bound declared for that subscript. This rule is included to increase agreement with X3.9-1978.

M - Never use more than three subscripts. This number is the maximum permitted by the ANSI standard X3.9-1966.

M - Do not use implicit indices for the first element of an array. For example, for a singly dimensioned array A, declared by

```
DIMENSION A(5), use
TEMP = A(1)

    not

TEMP = A
```

The lack of parentheses makes the program significantly less readable.

S - If the installation's compiler has an array subscript checking feature use it, at least during the initial testing phases of the program development. Since the automatic array subscript error checking feature tends to slow down a program considerably, it may be desirable to turn it off during production runs.

3.3.2 *Array Structure*

S - Avoid making assumptions regarding the sequential order in which array elements are stored in memory. For example, avoid using data statements of the form

```
DIMENSION A(2,3)
DATA A/1.,2.,3.,6.,8.,9./
```

4. Expressions

4.1 Arithmetic Expressions

4.1.1 *Evaluation of Expressions*

M - Never assume that some finite value, e.g., 0.0, will be substituted for a division by 0.

M - Always check, if possible, for the case when the denominator of an expression is zero.

M - Do not use ambiguous statements such as N+FUNC(N)*N where FUNC(N) alters N. The results of such a statement depends upon the compiler scanning algorithm. If the original value of N is not stored in a temporary location, FUNC(N) may destroy it.

M - Make heavy usage of parentheses to improve program clarity. There is no penalty in modern compilers for use of unnecessary parentheses. Do not rely upon the left to right evaluation of arithmetic expressions. For example, write the expression A/B*C as (A/B)*C, even though it is defined this way according to the ANSI standard. Some individuals might think it means A/(B*C). Leave no possible doubt as to what the compiler will do.

S - Break long arithmetic expressions into several simpler expressions. That is, by using intermediate variables, break an assignment statement having a long arithmetic expression into a series of assignments each of which has a simpler expression. This will improve program readability. For example, instead of writing

```
A=(B*C)+(((3**D)/(-3))*X)+(Y*4)
```

write

```
P1=B*C
P2=X*((3**D)/(-3))
P3=Y*4
A=P1+P2+P3
```

L - Polynomials such as A*X**4+B*X**3+C*X**2+D*X+E should be programmed as E+X*(D+X*(C+X*(B+X*A))) when efficiency is important. The expanded form is preferable when efficiency is not important because it is more readable.

4.1.2 *Type of Arithmetic Expressions*

M - Mixed mode expressions must be avoided, e.g., do not use A+B/C+D/7+E/4+F instead of A+B/C+D/7.0+E/4.0+F, or RCUT + NORDER*PERD instead of RCUT + FLOATF(NORDER)*PERD. The ANSI standard does not permit mixed mode expressions.

4.1.3 *Exponentiation*

M - Integer expressions must be raised to only integer powers, never real powers.

M - Do not use double exponentiation without parentheses, e.g., A**B**C. This form is not defined in the standard, so it may be implemented as A**(B**C), or (A**B)**C. If it is necessary to use double exponentiations, use parentheses to explicitly define the meaning intended.

4.2 Relational Expressions

The relational operators are: .GT. (greater than), .GE. (greater than or equal to), .LT. (less than), .LE. (less than or equal to), .EQ. (equal to), and .NE. (not equal to). Relational expressions have the form A1 OP A2, where A1 and A2 are arithmetic or masking expressions, and OP is a relational operator.

M - Relational operators must compare only integer expressions with integer expressions or real expressions with either real expressions or double precision expressions. Only these comparisons are permitted by the ANSI Standard X3.9-1966.

M - Do not use masking expressions with relational operators to form relational expressions. For example, do not use expressions such as (A .AND. B) .GT. (M .AND. .NOT. 77B). Masking expressions are not permitted by the ANSI X3.9-1966 standard.

M - Do not use relational operators for expressions of a COMPLEX data type. For example, do not use expressions such as AMT .LT. (1., 6.55). Use of expressions of the COMPLEX data type in this context is not defined by the ANSI X3.9-1966 standard.

M - Put at least one blank space on each side of a relational operator. to improve program readability, e.g., A .GT. B.

4.3 Logical Expressions

Logical expressions have the form L1 OP L2 OP L3...OP LN, where L1, ..., LN are logical operands or relational expressions and OP is a logical operator. The logical operators are .AND., .OR., and .NOT..

M - Do not use .N. for .NOT., .A. for .AND., or .O. for .OR. These abbreviations are not permitted by ANSI Standard 3.9-1966, and they make programs difficult to read.

M - Put a blank character on each side of a logical operator to improve program readability.

M - When an IF statement contains compound conditions, parenthesize the separate simple conditions to make the range and strength of the logical operators (.AND., .OR., and .NOT.) crystal clear. If there are more than two simple conditions, use continuation cards to put each continuation on a separate line and align conditions vertically.

S - Avoid unnecessarily complicated logical expressions, e.g.

```
IF (A .AND. B .OR. .NOT. C) GO TO 6
```

S - Avoid negative logical expressions whenever possible. Their positive equivalents are generally easier to understand. For example, instead of writing

```
      IF (.NOT. FLAG) GO TO 10
      X = Y
      GO TO 20
   10 CONTINUE
      A = B
   20 CONTINUE
```

write

```
      IF (FLAG) GO TO 10
      A = B
      GO TO 20
   10 CONTINUE
      X = Y
   20 CONTINUE
```

L - Use only logical variables or logical constants with logical operations. This approach is recommended only to achieve maximum portability of programs through compliance with X3.9-1966.

4.4 Masking Expressions

Masking expressions are similar to logical expressions, but the elements of the masking expression are of any data type (variable, constant, or expression) other than logical.

S - Avoid use of masking expressions (e.g., do not use: KAY .OR. 63, or .NOT. 55). This guideline is suggested because such expressions are not permitted by the ANSI X3.9-1966 standard, and also because they often depend upon the word-length of the computer. Both of these factors reduce program portability. This constraint is indeed unfortunate, since masking operations are extremely useful for bit-oriented operations, which are awkward to perform in FORTRAN in any other manner.

5. Assignment Statements

5.1 Arithmetic Assignment Statements

M - Do not use assignments of the form A=B, where (1) A is of the integer data type and B is of a complex data type, (2) A is real and B is complex, (3) A is double precision and B is complex, or (4) A is complex and B is anything but complex.

S - Avoid "run-on" equations. When possible, divide large equations into meaningful parts. This approach will help improve program clarity.

S - Do not use assignment statements requiring implicit conversion to REAL or INTEGER values. This should help avoid problems arising from rounding upon assignment to variables. For example, avoid expressions such as XI = I + 1 or I = J + 1.5

S - Put at least one blank character on either side of each equal (=) sign, to help improve program readability.

5.2 Logical Assignment

No recommended guidelines.

5.3 Masking Assignment

S - Avoid the use of assignments of the form V = masking expression. This type of assignment is not permitted by the ANSI standard, and the portability of programs is reduced by its use.

5.4 Multiple Assignment

M - Do not use assignments of the form $V = V_1 = V_2 = V_3 ... =$ expression. For example, do not use X = Y = Z = 4. This type of expression is not permitted by the ANSI standard. Also, its meaning can be ambiguous and depend upon the order in which it is evaluated.

6. Control Statements

6.1 GO TO Statement

M - Do not use the *GO TO statement to* achieve small gains in efficiency and thereby sacrifice program clarity.

M - Always put a blank character between the keywords GO and TO when using a GO TO statement. Also, put a blank to the right of TO and GO. For example, use GO TO 5, not GOTO 5.

6.1.1 Unconditional GO TO Statement

S - GO TO statements should be avoided wherever possible. Programs containing excessive GO TO's are inherently difficult to document and understand, since they tend to have spaghetti-like logic. If the GO TO statement is used, it should be used in a highly-controlled manner.

S - Where possible, use a DO loop, an IF statement, or a built-in function in lieu of a GO TO statement.

Example 1 (due to Ledgard and Chmura, see references)

Instead of:

```
10 CONTINUE
   IF (N .GT. M) GO TO 20
   . . . . . . .
   N = N + 1
   GO TO 10
20 CONTINUE
```

Use:

```
   DO 10 N = 1, M
   . . . . . . .
10 CONTINUE
```

Example 2

Instead of:

```
      IF (A .GT. B) GO TO 10
      C = B - A
      GO TO 20
   10 CONTINUE
      C = A - B
   20 CONTINUE
```

Use:

```
      C = A - B
      IF (C .LT. 0.) C = -C
```

Or, better still, use:

```
      C = ABS(A - B)
```

6.1.2 Computed GO TO Statement

M - Do not use an arithmetic or masking expression in a computed GO TO statement. Use only an integer variable. For example, do not use GO TO (10, 110, 11, 12, 13), X/Y. Use of the latter is not permitted by ANSI standard X3.9-1966.

M - The right parenthesis of a computed GO TO statement must always be followed by a comma to comply with X3.9-1966. For example, use

```
GO TO (10, 20, 30), L
```

not

```
GO TO (10, 20, 30) L
```

M - Do not assume that an incorrectly computed GO TO variable will result in a default condition such a "falling through." For example, do not use statements such as GO TO (100, 200, 300), M where M is 4.

M - If a flag has more than three values and is used for transfer of control to one of N locations, a computed GO TO statement is clearer than testing with multiple IF statements. For example, use

```
GO TO (10, 20, 30, 40, 50, 60), L
```

not

```
   IF (L-2) 10, 20, 25
25 CONTINUE
   IF (L-4) 30, 40, 45
45 CONTINUE
   IF (L-6) 50, 60, 65
65 CONTINUE
```

S - Check each computed GO TO statement for an out-of-bounds value of its index variable.

S - Avoid using the computed GO TO statement, except to simulate the CASE statement of structured programming as described in Appendix D.

6.1.3 ASSIGN Statement

M - Do not use the ASSIGN statement.

6.1.4 Assigned GO TO Statement

M - Do not use the assigned GO TO statement. It is a form of a program modifying itself, and it makes programs difficult to comprehend and debug. For example, when one sees an assigned GO TO statement in a listing, one has to make an extra effort to determine to where that statement transfers control. These disadvantages outweigh any small advantages that may be gained in CPU time and memory savings. Also it is always possible to do the same thing another way, e.g. using a computed GO TO statement or an IF statement.

6.2 Arithmetic IF Statement

6.2.1 Three-Branch Arithmetic IF Statement

M - Use a three-way branch IF statement only when the three branches are distinct, e.g., IF (A-80.) 500, 600, 700 is acceptable, but IF (A-80.) 500, 500, 700 is not. Instead use

```
IF (A .GT. 80.) GO TO 700
GO TO 500
```

M - The expression of an arithmetic IF statement must be either integer, real, or double precision.

S - The expression in an IF statement should explicitly include all levels of parentheses for clarity.

L - When testing a variable that can assume N possible values, include an (N+1)st check for the possibility that the variable has assumed an illegal value.

6.2.2 Two-Branch Arithmetic IF Statement

M - Do not use the two-branch IF statements, e.g., do not use IF (X*Y) 10, 20. It is not permitted by X3.9-1966.

6.3 Logical IF Statement

6.3.1 Standard Logical IF Statement

This statement has the form IF (eir) stat, where eir is a logical expression and stat is an unlabelled executable statement other than DO, END, or another standard-form logical IF statement.

S - Avoid testing floating-point variables for equality, since the results can be misleading and machine dependent. Due to the inherent inaccuracies of floating-point representations, it may happen that a test for a specific number will fail when the user would think it should pass. For example, A = 1.-3.0/3.0 may result in A being set to 0.0000001 and a subsequent test for (A .EQ. 0.) would fail. Instead of the test

```
IF (A .EQ. B) GO TO 2
```

use a statement like

```
IF ( ABS(A-B) .LE. 1.E-08) GO TO 2.
```

Tests for greater-than-or-equal-to or less-than-or-equal-to are preferred to tests for equality. This guideline is not a "must" because in some circumstances the value of the variable may have been previously set to an "exact" value.

S - In IF statements of the form IF (A .AND. B) (statement) do not assume that both A and B will be evaluated because, according to the ANSI Standard 3.9-1966, they don't have to be. For example, on the Univac 1108 system, if A is false, B is not checked for its status and the test fails. On the CDC 6600 system, both A and B are checked. As an example, if C is an array of size N and if core is preset to negative infinity, the statement

```
IF ((I .LT. N) .AND. (C(I) .LE.
C(I+1))) GO TO 100
```

would abort on the CDC machine if I = N when C(N+1) was evaluated, but it would not abort on the Univac system. This problem could be avoided on the CDC system by replacing it with

```
   IF(I .GE. N) GO TO 20
   IF(C(I) .LE. C(I+1)) GO TO 100
20 CONTINUE
```

L - If a few statements are to be executed only if some condition is met, it is a simple matter to set a logical value TRUE if the condition is true, and then use a series of logical IF's containing just that logical variable as the logical expression. The loss in computer time involved in repeating the test of the logical variable is very small in most cases, and will be more than compensated by increased clarity. This step helps avoid use of the GO TO statement.

<u>Example</u>

```
OK = .TRUE.
IF((I .GT. N) .OR. (I .LT. 1)) OK =
 .FALSE.
IF(OK ) A(I,J) = TEMP
IF(.NOT. OK) WRITE(6,9110) I
IF(.NOT. OK) NERROR = NERROR+1
```

6.3.2 Two-Branch Logical IF Statement

M - Do not use the two-branch logical IF statement. For example, do not use statements such as

```
IF ( K .EQ. 100) 60, 70.
```

This statement is not permitted by the ANSI X3.9-1966 standard.

6.4 DO Statement

6.4.1 DO Loops

M - DO loop variables must not be assigned a new value within the range of the DO-loop. Specifically, m_1, m_2, and m_3 should never be changed while executing a DO-loop of the form

DO n I = m_1, m_2, m_3

or

DO n I = m_1, m_2.

Altering a DO parameter within a loop may produce varying results, depending upon how the DO-loop feature was implemented in the compiler (pre-test, post-test, index in core, index in register, etc.). Changing of the DO-loop variables is not permitted by the ANSI standard. For example, do not write code such as

```
   DO 15  K = 1,10
      K = K + 1
      WRITE(5,2)K
15 CONTINUE
 2 FORMAT (I5)
```

M - Do not use nonpositive indices (I, m_1, m_2, m_3) in DO-loops. Loops in the form DO 10 I = K,J where J is less than K may or may not be executed once, depending upon where the test for completion is made. Non-zero values of these indices are not permitted by ANSI standard X3.9-1966. Although statements such as DO 10 I = 0,1 are permitted in Univac FORTRAN, they are not legal in CDC FORTRAN. Instead, replace such constructs with

```
DO 10 II = 1,2
   I = II-1
```

The effects of negative incrementing such as DO 10 I = NP,1,-1 can be achieved with the statements such as

```
DO 10 ID = 1,NP
   I = NP+1-ID
```

M - Do not use real variables as DO-loop indices. Although some machines, such as the Burroughs B5500, permit this feature, it is not allowed by the ANSI standard.

M - To improve program clarity, the executable statements in the range of a DO loop must be indented at least three spaces from the DO statement and terminating CONTINUE statement. Three spaces allow the "DO" to stand out, and nested loops will not deprive the programmer of too many card columns. For example, a DO-loop must appear as follows:

```
   DO 10 I = 1, NSHIPS
      Statement - 1
      Statement - 2
         ..
         ..
         ..
      Statement - n
10 CONTINUE
```

M - Do not assume that a DO-loop is always executed once.

M - Test for the case of zero iterations of a DO-loop, if this occurrence is a possibility. For example:

```
   IF((K-J) .LE. 0) GO TO 1
   DO 6 M = J,K
   . . .
   . . .
6 CONTINUE
1 CONTINUE
```

M - Do not assume that if m_1 is greater than m_2, the loop will be executed once. For example, the scope of DO 10 I=4, N where N=3 may not be executed once.

M - Always put a blank character between the keyword DO and the statement number following it, and between the statement number and the looping variable following it to improve clarity. For example, write DO 10 I = 1, 30, not DO10I=1,30.

M - Do not use large numbers as DO indices. Example: DO 10 I=1, N where N=2**17. The maximum size for a looping index must not exceed 2**15-1, i.e., 32767. Although the ANSI standard does not specify a value, this size is the maximum for some CDC FORTRAN compilers.

M - Every DO statement must refer to its own unique CONTINUE at the end of its range. Some compilers put special restrictions on nested DO-loops that terminate on one statement. The XDS Sigma 5/7, for instance, only allows the innermost DO to transfer directly to its termination point.

M - Do not assume a value of a DO index outside a DO loop if the loop terminated because the control variable is greater than its associated terminal parameter. In this case the ANSI Standard X3.9-1966 says its value is undefined. If the loop was exited by a GO TO statement or an arithmetic IF statement, i.e., by not satisfying the loop, the control variable is defined and is equal to the most recent value.

M - Do not transfer into a DO range. Although such a transfer was permitted by X3.9-1966, it is not permitted by X3.9-1978. The range of a DO loop may be entered only by the execution of a DO statement. Also, most compilers do not guarantee the results of such illegal transfers.

S - Invariant expressions should be factored out of DO loops to improve efficiency and program readability. For example, instead of

```
   DO 25 K = 1,30
      C = 3.0
      A(K) = C
25 CONTINUE
```

write

```
   C = 3.0
   DO 25 K = 1,30
      A(K) = C
25 CONTINUE
```

S - Parameterize DO loop indices rather than using literals whenever possible. For example, use DO 10 I = NL, NH instead of DO 10 I = 100, 325.

6.4.2 Nested DO Loops

M - Do not use the same single CONTINUE statement to terminate nested DO loops. Always end each DO Loop with its own unique CONTINUE statement. This convention should help isolate bodies of DO loops and thereby lead to a clearer code. For example, instead of

```
   DO 10 I=H,L
      DO 10 K=KL,KH
      . . .
10 CONTINUE
```

use

```
DO 10 I = H, L
   DO 20 K = KL, KH
   . . . .
   . . . .
20 CONTINUE
10 CONTINUE
```

M - DO loops must not be nested more than 25 deep. Although the ANSI standard does not have a limit, certain IBM FORTRAN compilers have 25 as an upper limit. As a practical matter and to preserve readability, DO loops should not be more than four deep.

M - The range of a contained DO must be a subset of the range of the containing DO.

6.5 CONTINUE Statement

M - Always end each DO loop with its own unique CONTINUE statement.

S - Use CONTINUE statements liberally to improve program clarity and to facilitate debugging.

S - Put blank comment statements before and after each *CONTINUE statement* that is a major decision point in program. Add additional comment statements to explain the decision being made.

6.6 PAUSE Statement

M - Do not use the PAUSE statement. Some computer facilities do not permit its use, even though it is permitted by ANSI Standard X3.9-1966.

6.7 STOP Statement

M - Do not use the form of the stop statement, STOP ≠C...C≠, where C...C is a string of characters. Use only the simple four-character STOP, or STOP n, where n is an actual digit string of length one to five. The character form is not permitted by ANSI Standard X3.9-1966.

6.8 END Statement

M - Do not put a statement label (number) on an END statement. Such a label is not permitted by the ANSI standard.

M - Every program must physically terminate with an END statement.

6.9 RETURN Statement

M - Do not use return statements of the form RETURN i. This form is not permitted by the ANSI standard. Use only the six-character form, RETURN.

M - Do not use a RETURN statement in the main program. A RETURN statement may appear only in a procedure subprogram, according to ANSI Standard 3.9-1966.

M - Use at most one RETURN statement per subroutine. Although the use of more than one RETURN is permitted by the ANSI standard, it can lead to poorly structured programs, since it defeats the one-in one-out principle of structured programming.

M - A RETURN statement must be used only as the last executable statement of a subroutine or function. This convention will help produce one-in/one-out control structures. Some machines permit the END statement to act also as a normal RETURN statement, but other machines require at least one RETURN statement before the END. Always include the RETURN statement.

7. Specification Statements

M - Keep all specification statements at the beginning of routines (programs, subroutines, or functions).

M - When used, specification statements must appear in the following order:

TYPE
DIMENSION
COMMON
EQUIVALENCE (to be avoided, if possible; see Section 7.4)

M - Do not scatter specification statements. Group all DIMENSION statements together. Similarly, group COMMON blocks and DATA statements together. Incremental compilers cannot handle scattered type, DIMENSION, and DATA statements; yet this form of compiler is desirable from a speed/user interface standpoint.

7.1 Type Statements

7.1.1 EXPLICIT Type Statements

M - The only data types permitted are INTEGER, REAL, DOUBLE PRECISION, COMPLEX and LOGICAL

M - The standard naming conventions, namely, using the letters A through H and O through Z as the first characters of real variables, and I through N as the first characters of integer variables, must be used and not overridden. For example, do not use INTEGER SUM, A, B, or REAL ZERO.

M - Always include the word PRECISION in DOUBLE PRECISION type statements. For example, instead of using DOUBLE ALIST, J, B, use DOUBLE PRECISION ALIST, J, B. This statement is required by ANSI standard X3.9-1966, and also improves program clarity.

M - Do not specify the type of a name more than once in a program unit, as per X3.9-1978.

7.1.2 IMPLICIT Type Statements

M - IMPLICIT type statements must not be used. For example, do not use IMPLICIT INTEGER (A-F,H). These statements are not permitted by the ANSI standard 3.9-1966. They also reduce program readability.

7.2 DIMENSION Statement

M - Do not use more than three dimensions in an array. For example, do not use DIMENSION A(10,20,30,5). The use of more than three dimensions is not permitted by ANSI standard 3.9-1966.

M - If an array appears in a COMMON area, it must be dimensioned within the COMMON block and not in a DIMENSION declaration, e.g., use

```
COMMON/INK/A(100), B
```

not

```
DIMENSION A(100)
COMMON/INK/A,B
```

This guideline improves clarity and conciseness.

M - Do not use adjustable dimensions in the main program. They may appear only in procedure subprograms, according to ANSI X3.9-1966. For example, DIMENSION A(L,K,M) is permitted in the main program of Univac 1108 FORTRAN programs, with values set by parameter cards. It is not permitted, however, in CDC FORTRAN.

M - Adjustable dimensions must only be integer variables.

M - In a subprogram, a symbolic name that appears in a COMMON statement must not identify an adjustable array.

L - Group dimensioned variables in alphabetical order under one dimension declaration. This grouping can improve program clarity by making it easier to find the variables.

7.3 COMMON Statement

M - Numbered COMMON must not be used. It is not permitted by the ANSI standard X3.9-1966.

M - COMMON block names must not exceed six characters.

M - A given COMMON block must have the same number of variables, and each variable must have the same number of elements, independent of the routine in which the common block appears.

M - Do not declare a COMMON block name more than once in a COMMON statement or program unit.

M - Corresponding variables in COMMON blocks must use the same names in all routines.

M - Include a COMMON area in a subroutine only if it is used in that subroutine. Following this guideline will improve program readability.

M - Do not use more than 60 COMMON blocks.

M - Do not use blank COMMON unless absolutely necessary. If necessary, lay out blank COMMON in one central routine and treat variables there as if they were global. This procedure assists in avoiding duplication and is a form of documentation

S - Avoid excessive use of labelled COMMON. (Insufficient blank COMMON may result in inability to load in OS/360 due to the loader sharing that space.)

S - In specifying COMMON block names, leave space for six characters, e.g., COMMON/LINK1 /I,J, A(100). This guideline will simplify program modifiations should it be necessary to change a block name.

S - When using COMMON be careful to avoid the hazards of context effects.

L - Group associated variables in a single COMMON area. Data types having the greatest word length requirements should appear first. Within each type of variable, arrays should appear last. This grouping helps make programs more readable.

7.4 EQUIVALENCE Statement

S - Avoid using the EQUIVALENCE statement, except when absolutely necessary to save storage space. Although permitted by the ANSI standard, this statement tends to make programs less readable.

M - Always include subscripts when arrays are equivalenced. For example, do not assume

```
DIMENSION ZEBRA(10)
EQUIVALENCE (ZEBRA, TIGER)
```

means the same as

```
DIMENSION ZEBRA(10)
EQUIVALENCE (ZEBRA(1), TIGER(1))
```

M - The number of subscript expressions of an array element name must correspond in number to the dimensioning of the array or declarator, in accordance with ANSI Standard X3.9-1978.

M - The EQUIVALENCE statement is used to permit the sharing of storage by two or more entities. Do not use it to equate mathematically two or more entities.

L - Although INTEGER and REAL variables should never needlessly be equivalenced to each other, there are some instances when this is very valuable, such as when creating data structures. In general, however, avoid declarations such as EQUIVALENCE (A,I), since they severely reduce program readability.

7.5 LEVEL Statement

M - Do not use LEVEL statements. They are not permitted by ANSI standard X3.9-1966.

7.6 EXTERNAL Statement

M - If an external procedure name is used as an argument to another external procedure, it must appear in an EXTERNAL statement in the program unit in which it is so used, in accordance with ANSI X3.9-1966.

S - Avoid EXTERNAL statements whenever possible. Although permitted by ANSI 3.9-1966, they are confusing to many programmers.

7.7 DATA Statement

M - Do not use parentheses in DATA statements. For example, do not use DATA (A = 3.), (B = 4.115). Instead, use DATA A, B/3.,4.115/.

This is an unfortunate rule since the first form is inherrently clearer than the second. The reason for it is that the parenthesized form is not permitted by the ANSI standard. To comply with the ANSI standard, DATA statements must have only the form DATA $Vlist_1/Dlist_1/,\ldots Vist_n/Dlist_n/$ where,

Vlist = a list of array elements or variable names, separated by commas. Array elements must have integer constant subscripts.

Dlist = a list of one or more of the following forms, separated by commas: a constant or rf*constant, where rf is an integer constant. The constant is repeated rf times.

M - The number of elements in the Vlist must equal the number in the corresponding Dlist.

M - The type of the constant in the Dlist must agree with the type associated with the corresponding name in the Vlist.

M - Do not use an implied DO in a DATA statement. For example, do not use DATA (A(I), I=1,10)/1.,2.,3., 7*2.5/. Unfortunately, the implied loop is not permitted by the ANSI standard. It is, however, far superior to just using the array name (especially for two or three dimensional arrays). For example, the implied loop is easier to read and write, and is less error prone than the following:

```
DATA A /1.,2.,3.,2.5,2.5,2.5,
       2.5,2.5,2.5,2. 5/
```

Unfortunately, the ANSI recommendations are:

```
  DATA A(1), A(2), A(3), A(4), A(5),
+      A(6), A(7), A(8), A(9),
+      A(10)/1.,2.,3.,2.5,2.5,2.5
+            2.5,2.5,2.5,2.5/
```

or

```
  DATA A(1)/1./,A(2)/2./,A(3)/3./,
+      A(4)/2.5/,A(5)/2.5/,A(6)/2.5/,
+      A(7)/2.5/,A(8)/2.5/,A(9)/2.5/,
+      A(10)/2.5/
```

M - An initially defined variable or array element may not be in blank common, according to X3.9-1966.

M - A variable or array element in a labeled COMMON block may be initially defined in only a block data subprogram, in accordance with X3.9-1966.

S - Avoid defining the value of the same variable in several places throughout a program. For example, avoid setting the variable PI = 3.14159 in several subroutines. It would be better to initialize this variable once, and pass it to other routines via a COMMON block.

8. Programs, Subprograms, and Procedures

A program unit consists of a set of FORTRAN statements, with comments, followed by an END card. A main program is a program unit that does not begin with a SUBROUTINE, FUNCTION, or BLOCK DATA statement. It can be used as a self-contained computing procedure. A subprogram is a program unit that begins with SUBROUTINE, FUNCTION, or a BLOCK DATA statement.

8.1 Main Programs

M - The beginning of the text of the main program must describe, in comment cards, the following:

(1) - The purpose of the program.

(2) - The author(s) name, address, organization, and phone number.

(3) - The version number of the program.

(4) - The date of the first program compilation.

(5) - The date the program was last updated.

(6) - The organization for which the program was written.

(7) - The processing performed by the program.

(8) - A listing of external reports, books, or other documents describing the algorithms used, or other information about the program.

(9) - A list of COMMON block variables modified by the main program.

(10) - A description of the card input required by the program (optional).

(11) - The names and contents (briefly described) of all files (tape or disk) written and/or read by the program.

(12) - The names of subroutines in which the above files are read or written (optional).

(13) - A description of the output produced by the program (optional).

(14) - A list of "options" available in the program (optional).

(15) - A list of changes made to the program and dates of those changes (optional).

M - The main program must have no more than 50 executable statements. Longer programs are generally difficult to understand and maintain.

M - If an entity of a given common block is given an initial value in a BLOCK DATA subprogram, a complete set of specification statements for the entire block must be included. This is required by X3.9-1966.

S - Use a top-down approach when designing programs.

8.2 Block Data Subprogram

M - Do not use BLOCK DATA subroutines that have names. They are not permitted by ANSI Standard X3.9-1966. Use only unnamed BLOCK DATA statements.

M - A program must contain no more than one BLOCK DATA subprogram. (In compliance with X3.9-1978).

8.3 Procedures

M - Make frequent use of functions and subprograms to clarify and modularize the source code.

M - Main programs, subroutines, and functions should be kept to a minimum number of lines. They must have no more than 50 executable statements. This constraint helps to make programs more understandable by reducing the need for the programmer to keep in mind the actions of large blocks of code.

M - At the beginning of each procedure, a blank comment statement must be followed by a set of comment statements which describe what the procedure does and the meaning of each of the formal parameters. Parameters must be identified as to whether they are input, output, or input-output variables.

M - At the beginning of each procedure, there must be a set of comments indicating which common block variables are modified by this subroutine. These comments are needed to facilitate maintenance of the program and to avoid wasting time searching through cross reference lists.

M - Procedures must be arranged in the source code in alphabetic order following the main program. This arrangement makes programs significantly easier to debug and understand, since it cuts down on time searching for subroutines in large printouts.

M - Procedures which are never called must be eliminated from the source deck. Extraneous routines waste others' time trying to determine their purpose and relevance. They also waste memory space.

M - Always put at least one blank character after the keyword CALL.

M - Do not use recursive procedures. Most compilers do not allow them.

S - Procedures should describe, in comment statements at the beginning of each routine, the meaning of internal variables used in the routine.

S - Sections of code likely to change in the future should be isolated into procedures and clearly identified whenever possible.

S - Make procedures general purpose whenever possible.

L - The comment-statement list of descriptions of the formal parameters of a procedure should appear in alphabetic order, thereby making it easier to find the description of a given variable.

8.3.1 *SUBROUTINE Subprograms* (also see Section 6.9, RETURN statement.)

M - Do not use subroutine calls that include a return list; for example, CALL PGMI(A,B,C), RETURNS (5,10). Instead use the simple RETURN statement. The return list is not permitted by ANSI standard 3.9-1966.

M - Do not use the RETURN i form of returning from a subroutine. Use the simple RETURN statement. The RETURN i form is not permitted by the ANSI standard X3.9-1966.

M - The symbolic name of the formal parameters of a subroutine subprogram must not appear in an EQUIVALENCE, COMMON, or DATA statement in the subprogram.

M - The symbolic name of a subroutine must not appear in any statement in in the subprogram except in the symbolic name of the subroutine itself. This is required by ANSI X3.9-1966.

M - Do not use a subroutine when a function is needed. Use the right tool for the job.

M - Do not use the ENTRY statement. Although its use is permitted by the ANSI standards, it defeats the one-in/one-out principle of structured programming, since it allows more than one entrance into a program unit. Also it is annoying to search code looking for it, i.e., it makes a program less readable.

M - Do not use any language feature which permits subprograms defined within other programs to have access to all parent program variables. Some compilers, such as the XDS Sigma 7, allow this to occur. The use of this feature is not permitted by the ANSI standards.

8.3.2 *FUNCTION Subprograms*

M - Do not declare double-precision type functions with just the identifier DOUBLE; always use DOUBLE PRECISION. The use of DOUBLE alone is not permitted by the ANSI standard.

M - Always include a RETURN statement in a function, do not assume just an END statement will suffice.

M - The formal parameters of a function must not be assigned new values within the body of the function. That is, there must not be any input-output or output formal parameters; only input parameters are permitted. If formal parameters must be changed, a subroutine must be used.

M - Do not use a function when you need a subroutine.

M - Do not alter COMMON block variables in a function.

8.3.3 *Basic External Functions*

M - Appendix B lists the functions required by X3.9-1966. Avoid using any other external function, other than TAN. It is unfortunate TAN, the tangent function, was not included in X3.9-1966.

S - Avoid using the external functions SINH, DSINH, COSH, COSH, DCOSH, ACOS, ASIN, DTANH, DTAN, CDABS. Some systems may not recognize these routines.

8.3.4 Intrinsic Functions

M - Appendix C lists the intrinsic functions allowed by X3.9-1966. Do not use other intrinsic functions.

S - Avoid using the following intrinsic functions since some systems may not support them:

logical product	AND(X,Y,Z)
logical sum	OR(X,Y,Z)
exclusive or	XOR(X,Y,Z)
complement	COMPL(A)
shifting	SHIFT(A,I)
masking	MASK(I)
random number	RANF(A)
location of variable	LOCF(Q)

Shifting algorithms are usually word-size and hardware dependent. If it is necessary to use such routines, include comment statements describing their purpose.

8.3.5 Additional Utility Subprograms

S - Avoid all operating system interface routines, such as calls to DATE, TIME, SENSE SWITCH settings, overlays, and recovery routines. These routines tend to significantly reduce the portability of programs among different computers. If they are used, clearly describe their function in comment cards.

S - Avoid embedding system-dependent debugging aids in programs.

S - Avoid using system-dependent calls to random number generators.

S - Avoid calls to system-dependent mass storage I/O routines.

S - Avoid calls to system-dependent routines that check for end-of-file, or parity errors.

S - Avoid calls to other system-dependent I/O routines, such as those that give information on size of last buffer read in, or that define tape labels.

S - Avoid calls to routines that manipulate the transfer of data to and from extended (ECS) or large (LCS) core storage.

S - Avoid calls to routines that handle terminal I/O.

8.3.6 Statement Functions

M - Do not use statement functions that include masking expressions.

M - Aside from dummy arguments, the expression of a statement function may only contain non-Hollerith constants, variable references, intrinsic function references, references to previously defined statement functions, and external function references.

8.3.7 Procedure Communication

M - Do not use more than 60 formal parameters in each procedure call. The ANSI standard does not specify any limit. Some CDC compilers, however, have a limit of 60 parameters.

M - With the exception of a Hollerith constant, the actual arguments in a subroutine call must agree in type and number with the corresponding formal parameters in the subroutine, e.g., a call to a subroutine using CALL SUB1(1,1.0) must be avoided when the subroutine begins as subroutine SUB1(A,I). The Hollerith constant is an exception to the rule regarding agreement of type.

M - Literals must never be used as arguments in subroutine calls when their corresponding formal parameters can be changed in the called routine.

M - Do not use multiple entry points into a routine. Each subroutine and function must have only one entry point. Use of more than one entry point defeats the one-in/one-out structured-programming concept.

M - Do not use variable-length argument strings in procedure calls. Some compilers will not deal successfully with missing arguments (variable length) in a subroutine CALL.

M - The formal parameters of a function must not be assigned new values within the body of the function.

M - Do not assume subroutines will be called with correct arguments.

S - The actual parameters of a subroutine should be listed so that input parameters are given first, input-output parameters are given second, and output parameters are specified last. This listing order should help improve the readability of the programs.

S - Calling sequences should be used as little as possible. COMMON is a much more efficient method of communication between program units.

S - Calls to machine-dependent subroutines should be avoided.

S - Parameters should always be checked for validity when read from cards, files, or upon entering subroutines. The intent here is to detect input errors as early during program execution as possible.

9. Input/Output

9.1 FORTRAN Record Length

M - Logical record lengths must not exceed 80 characters.

S - The record length for print files should not exceed 120 characters. There are some circumstances, however, such as when making line printer plots, where the additional length is necessary.

9.2 Carriage Control

M - Use separate field specification for printer control, e.g., use FORMAT (1H1,7HBUFFER=) instead of FORMAT (8H1BUFFER=).

9.3 READ and WRITE Statements

M - Do not use PRINT statements. These are not permitted by the ANSI standard X3.9-1966.

M - Do not use PUNCH statements. These are not permitted by ANSI standard X3.9-1966.

M - A simple I/O list enclosed in parentheses is prohibited from appearing in an I/O list (in compliance with X3.9-1978).

M - Do not use expressions for unit numbers, e.g., READ(2*K+1,10). They are not permitted by the ANSI standard X3.9-1966. The unit number must be either an integer variable or an integer constant.

M - Assume the users of a program will provide it with bad input data. Always check these data for validity.

M - All printed output must be annotated so that it is understandable to a user who does not understand the inner workings of the program.

S - All input parameters should be checked very closely for proper values. Parameters should be printed out with an identifying label as soon after input as possible, to facilitate debugging.

S - Avoid operator interaction (typewriter I/O). Some installations may not support this feature.

S - Avoid using alternative action flags; for example, READ (N,ERR=101,END=122). These are not permitted by the ANSI standard. This is not a mandatory requirement because with some FORTRAN compilers (e.g., Univac 1108) they are needed to check for an end-of-file, and to perform other file operations.

S - Check input parameters for reasonableness and validity as soon as possible after they have been read in. Avoid beginning calculations without first checking the inputs.

S - Incorporate a debug switch in the program which will print out useful trace information. This switch should be designed so that it can be turned on and off after the program is compiled (i.e., at execution time).

S - Make use of as many operating system checks on tape labels as possible. By checking tape labels in the software, one can often avoid disasterous mistakes.

9.3.1 Formatted

M - For formatted I/O, use only write statements of the form WRITE (un,fn) iolist and READ statements of the form READ(un,fn) iolist. Here un and fn identify the input/output unit and format specification, respectively.

M - I/O device numbers must not be negative. Negative numbers were permitted by X3.9-1966, but not X3.9-1978.

S - I/O device numbers (un) often depend upon the system. They should, therefore, be referred to symbolically, e.g., NREAD, NWRITE, rather than by literals, such as 5 and 6. This practice facilitates moving the programs to another machine and also enables the user to easily modify his program to output to a private file, instead of the system output file, if he so wishes. Only unsubscripted integer variables should be used for I/O device numbers.

9.3.2 Unformatted

M - For unformatted I/O, use only the form READ(u) iolist or WRITE(u) iolist to comply with ANSI standard X3.9-1966.

S - Design magnetic tape outputs for general compatability with other machines. Avoid complicated blocking or binary (non-formatted) file outputs.

9.4 FORMAT Statements

M - REAL constants must never be read from cards with INTEGER formats and vice versa.

M - Do not use octal or hexadecimal specifications in FORMAT statements, since they are not permitted by ANSI standard X3.9-1966.

M - In each program module, all FORMAT statements must be listed after all executable statements. This makes programs easier to debug and read.

M - Do not use list-directed (free-field input) I/O statements. For example, do not use statements such as READ(U,*) iolist or READ *,iolist. These are not permitted by ANSI standard X3.9-1966.

M - Do not perform alphanumeric conversion of the form rRw.

M - Put a comma after each field (except groups of more than one slash (/)) in a FORMAT statement, even though the compiler may accept the statement without the commas.

These commas make FORMAT statements more readable, and some compilers require their presence. For example, do not write

```
9510 FORMAT (F10.0,9XL1).
```

Instead, write

```
9510 FORMAT (F10.0, 9X, L1)
```

M - Put at least one blank space after the comma following each field in a FORMAT statement. This will make it easier to read and modify formats in the future.

M - When formatted records are prepared for printing, the first character of the record is not printed. The first character determines vertical spacing as follows: blank-one line, 0-two lines, 1 to the first line of the next page, +-no advance. These are the only control characters allowed by ANSI X3.9-1966, and only these can be used.

M - Do not use format-controlled records of more than 120 characters.

S - When performing alphanumeric conversions in the form rAw, r should be no greater than 4. This is because we can only assume that machines will have at a minimum the ability to pack four characters per computer word.

S - Format statement fields after slashes(/) should begin on a new line. That is, the record terminator "/" that appears in a format statement should mark the end of the FORTRAN text as it appears on a line of the source listing (except for a succeeding comma). Thus the end of a line on the printer output will correspond to the end of a line on the FORTRAN source listing, e.g., use

```
      FORMAT (16H NO. OF FREQS = , I3 ,/,
    1       17H NO. OF DEPTHS = , I3 )
```

not

```
      FORMAT(16H NO. OF FREQS = ,I3,/,17H
    1  NO. OF DEPTHS = ,I3).
```

S - In FORMAT statements use the specification form NH----- instead of '------' . For example, use 4HABCD, not 'ABCD' (or *ABCD*). Although the apostrophe notation is a tremendous timesaver for the programmer since it alleviates the need to count characters (which is highly error prone), it is not permitted by ANSI standard X3.9-1966. It is permitted by X3.9-1978 and is generally supported in one form or another on most compilers. For maximum portability, it should be avoided.

S - Repeated specifications in FORMAT statements should not be more than two deep. For example, avoid statements like 9110 FORMAT (1X, 3(I5, 2(1X,I3,3(I4, 2X)))).

S - Separate multiple card format statements by blank comment statements.

L - Following the E or D in an E or D output field, a + or - should be used prior to the exponent. This increases compliance with ANSI X3.9-1978. ANSI X3.9-1966 permitted a blank as a replacement for a +.

L - All format statement numbers should begin with 9 and have four digits.

L - Group all input format statements together and precede with a comment statement with the word "INPUT". Put a comment card, with all asterisks before and after this card, to make it stand out.

L - Group all output format statements together and precede with the word "OUTPUT". Add asterisk comment cards as described above.

L - Group all [illegible] used for both input and output together and precede with a comment statement with the word "INPUT/OUTPUT". Add asterisk comment cards as described above.

L - Group all error message format statements together and precede with the words "ERROR MESSAGES". Add asterisk comment cards as described above.

9.5 File Manipulation Statements

M - A record must not be written after an end of file record in a sequential file. X3.9-1966 does not prohibit this, but X3.9-1978 does.

M - A sequential file must not contain both formatted and unformatted records.

Mixing of the two is permitted by X3.9-1966, but not X3.9-1978.

S - The use of overlays should be avoided (whenever possible). They are not permitted by X3.9-1966.

S - The use of segmentation should be avoided (whenever possible). It is not permitted by X3.9-1966.

S - Avoid special disc or drum-oriented instructions. They are not standard forms. If necessary, be sure they are well-isolated and clearly identified with comments.

S - Avoid making assumptions regarding number and kind of peripherals available.

S - Isolate and clearly mark code that checks for end-of-files. This practice should help reduce coding changes necessary to transfer the program to another computer. Such statements should be avoided whenever possible, because of their machine dependence.

9.6 BUFFER Statements

M - Do not use BUFFER statements. They are not permitted by the ANSI standard X3.9-1966.

9.7 NAMELIST

M - Do not use the NAMELIST capability. This is not permitted by ANSI standard X3.9-1966.

9.8 ENCODE and DECODE

M - Do not use the ENCODE and DECODE facilities. Their use is not permitted by ANSI standard X3.9-1966.

10. Miscellaneous Machine/System Dependencies

M - Do not assume that memory will be zeroed before the program runs.

M - Whenever a variable has a chance of being used without initialization, on another system, always explicitly initialize memory to zero, even if the system presently being used does it for you. For example,

```
C  ** CLEAR ARRAYS
   DO 20 I = 1, 10
      X(I) = 0.0
      DO 10 J = 1,81
      A(I,J) = 0.0
10    CONTINUE
20 CONTINUE
```

M - Machine-dependent code that cannot be eliminated must be isolated and clearly identified with comment cards.

M - Programs must be modularized into machine/system dependent and independent sections.

M - Do not use any code, such as sorting, that depends upon the internal representation of characters.

M - Do not use hex or decimal and octal literals (an example of internal representations).

M - Do not write code that depends upon BCD, or EBCDIC card code differences.

M - Do not use word, byte, character, or system implementation-dependent coding.

M - Always assume the character set will be different for different machines. Do not make programs dependent upon the internal character representation of a particular machine.

M - Do not use programming tricks dependent upon machine idiosyncrasies.

M - Write your programs so the machine operator can use his time and talents efficiently. For example, don't require him to set sense switches or needlessly mount and dismount tapes.

M - Make your programs as operator-proof as possible. Don't have a program ask the operator for information if this same information can be obtained from the operating system another way. For example, don't require him to type in the date.

S - Avoid assembly language interfaces. Their use is not permitted by the ANSI standard.

S - When writing programs, estimate the range of values variables can take and document the same. The precision of integer and floating-point arithmetic is machine and software dependent. If future systems have fewer bits assigned to the characteristic in floating-point representations, for example, the current data may generate over/underflows (which may go undetected).

S - Never assume the computer operator has done everything correctly.

S - Avoid whenever possible multitasking statements, e.g., statements such as ATTACH and DELETE in System/360, the FORK statement in the XDS-940, and the ZIP statement in the Burroughs B5500. The structure of most multitasking programs is very complex and difficult to debug.

11. Summary of Fortran Statements and Recommendations

The following is a list of FORTRAN statements and recommendations regarding their use. This list contains statements which must not be used under any circumstance (--), which can be used only when necessary (-), which can be used at will (+), and which are highly recommended (++).

The notation used here is as follows:

V = variable
Sn = statement number
iv = integer variable
m_1, m_2, m_3 = integer constants
n = integer

Section No.		Rating
See 5.1	Assignment Statements	
See 5.2	V = arithmetic expression	+
See 5.3	V = masking expression	--
See 5.4	Multiple Assignment	
	$V = V_1 = V_2 --- = V_n =$ expression	--
	Control Statements	
See 6.1.1	GO TO Sn	-
See 6.1.2	GO TO (Sn_1, Sn_2---Sn_m), iv	-
See 6.1.2	GO TO (Sn_1, Sn_2---Sn_m), expression	--
See 6.1.4	GO TO iv (Sn_1, Sn_2---Sn_m)	--
See 6.1.4	GO TO iv (Sn_1,Sn_2---Sn_m)	--
See 6.1.3	ASSIGN Sn to iv	--
See 6.2.1	IF (arithmetic exp) Sn_1, Sn_2, Sn_3	+
See 6.2.1	IF (masking exp) Sn_1, Sn_2, Sn_3	--
See 6.2.2	IF (arithmetic or masking exp) Sn_1, Sn_2	--
See 6.3.1	IF (logical expression or relational exp) stat	+
See 6.3.2	IF (logical express or relational exp) Sn_1, Sn_2	--
See 6.4	DO Sn iv=m_1, m_2, m_3	+
See 6.4	DO Sn iv=m_1, m_2	+
See 6.5	CONTINUE	++
See 6.6	PAUSE	--
See 6.6	PAUSE n	--
See 6.6	PAUSE ≠c...c≠	--
See 6.7	STOP	-
See 6.7	STOP n	-
See 6.7	STOP ≠c...c≠	--
See 6.8	END	+
See 7.1	Type Declaration	
	INTEGER $name_1$,....,$name_n$	--
	TYPE INTEGER $name_1$,....$name_n$	--
	REAL $name_1$,....,$name_n$	--
	TYPE REAL $name_1$,....,$name_n$	--
	COMPLEX $name_1$,...,$name_n$	+
	TYPE COMPLEX $name_1$,....$name_n$	--
	DOUBLE PRECISION $name_1$,.... $name_n$	+
	DOUBLE $name_1$,---$name_n$	--
	TYPE DOUBLE PRECISION $name_1$,...$name_n$	--
	TYPE DOUBLE $name_1$,...$name_n$	--
	LOGICAL $name_1$,...$name_n$	+
	TYPE LOGICAL $name_1$,...$name_n$	--
	IMPLICIT type (ac),...$type_n$(ac)	--
See 7.6	Declaration	
	EXTERNAL $name_1$,...$name_n$	-

Storage Allocation

See 7.1.1	type $name_1$, (di)	--
See 7.1.1	TYPE type name (di)	--
See 7.2	DIMENSION $name_1$ (d_1)...$name_n$ (dn)	+
	di array declarator, one to three integer constants; or if name is a dummy argument in a subprogram, one to three integer variables or constants	
See 7.3	COMMON V_1,...V_n	-
See 7.3	COMMON /blk name/V_1....,V_n	+
See 7.3	COMMON // V_1, V_2,...V_n	-
	where blk name symbolic name or	+
	1-7 digits	--
	// blank common	-
See 7.4	EQUIVALENCE ($glist_1$,),...($glist_n$)	-
See 7.5	LEVEL n,a_1...a_n	--
See 7.7	Data $Vlist_1$/$dlist_1$/.. $Vlist_n$/$dlist_n$/	+
See 7.7	Data ($Vlist_1$ + $dlist_1$),...($Vlist_n$ + $dlist_n$)	--
	$Vlist_1$ List of array elements, variable names, separated by commas	+
	List of array names,	--
	implied DO list	-
	dlist One or more of the following forms separated by commas:	
	constant	+
	rf* constant	+
	(constant list)	--
	rf* (constant list)	--
	constant list: list of constants separated by commas	
	rf: integer constant, the constant or constant list is repeated the number of times indicated by rf	

See 8.1 Main Programs

	PROGRAM Name	+
	PROGRAM name (par_1,...par_k)	-

See 8.3 Subprograms

See 8.3.2	Function name (p_1,...p_n)	++
See 8.3.2	type FUNCTION name (p_1,...p_n)where type is	
	COMPLEX, DOUBLE PRECISION, LOGICAL	++
	where type is DOUBLE, INTEGER, REAL	--
See 8.3.1	SUBROUTINE name (p_1,...p_n)	++
See 8.3.1	SUBROUTINE name	++
See 8.3.1	SUBROUTINE name (p_1,...p_n) returns (b_1,..b_m)	--
See 8.3.1	SUBROUTINE name, RETURNS (b_1,...b_m)	--
See 8.3.1	ENTRY name	--

Statement Functions

See 8.3.6	name ($p_1, \ldots p_n$) = expression	+

Subprogram Control Statements

See 8.3.7	CALL name	+
See 8.3.7	CALL name ($p_1, \ldots p_n$)	+
See 8.3.7	CALL name ($p_1, \ldots p_n$) RETURNS ($b_1, \ldots b_m$)	--
See 8.3.7	CALL name, RETURNS ($b_1, \ldots b_m$)	--
See 6.9	RETURN	+
See 6.9	RETURN i	--
See 8.2	BLOCK DATA	--
See 8.2	BLOCK DATA name	--

Input/Output

See 9.3	PRINT anything	--
See 9.3	PUNCH anything	--
See 9.3.1	WRITE (u,fn) Vlist	+
See 9.3.1	WRITE (u,fn)	+
	WRITE fn, Vlist	--
	WRITE fn	--
See 9.3.2	WRITE (u) iolist	-
See 9.3.2	WRITE (u)	-
	WRITE (w,*) iolist	--
	WRITE *, iolist	--
See 9.3.1	READ (u,fn) iolist	+
See 9.3.1	READ (u,fn)	+
	READ fn, iolist	--
See 9.3.2	READ (u) iolist	-
See 9.3.2	READ (u)	-
See 9.3	READ (u,*) iolist	--
See 9.3	READ *, iolist	--
See 9.6	BUFFER IN (n, p) (a, b)	--
See 9.6	BUFFER OUT (u, p) (a, b)	--
See 9.7	NAMELIST /group name/$a_1, \ldots$an/group $name_n$/ $a_1, \ldots$an/	--
	READ(u, group name)	--
	WRITE(u, group name)	--

Internal Transfer of Data

See 9.8

	ENCODE (c, fn, v) iolist	--
	DECODE (c, fn, v) iolist	--

File Manipulation

	REWIND u	+
	BACKSPACE u	+
	ENDFILE u	+
	EOF(U)	-

Format Specification

	S_n FORMAT ($fs_1, \ldots fs_n$)	+
	fsi one or more field specifications separated by commas and/or grouped by parentheses	

Data Conversion

	srEw.d	Single precision floating-point with exponent	+
	srEw.dEe	Floating point with specified exponent length	--
	srEw.dDe	Floating point with specified exponent length	--
	srFw.d	Single-precision floating-point without exponent	+
	srGw.d	Single-precision floating-point with or without exponent	+
	srDw.d	Double-precision floating-point with exponent	+
	rIw	Decimal integer conversion	+
	rIw.z	Integer with specified minimum digits	--
	rLw	Logical conversion	+
	rAw	alphanumeric conversion	+
	rRw	alphanumeric conversion	--
	rOw	Octal integer conversion	--
	rOw.z	Octal with conversion with minimum number	--
	rZw	Hexadecimal conversion	--
	SrVw.d	Variable type conversion	--
	s	Optional scale factor of from: nP	+
	r	Optional repetition factor, non-zero unsigned integer	+
	w	Integer constant indicating field width	+
	d	Integer constant indicating digits to right of decimal point	+
	e	Integer indicating digits in exponent field	--
	z	Integer specifying minimum number of digits	--
	nX	Intraline spacing	+
See 9.4	nH	Hollerith	+
See 9.4	*...*	Hollerith	--
	≠...≠	Hollerith	--
	'...'	Hollerith	-
	/	Format separator; indicates end of FORTRAN record	+
	Tn	Column tabulation	--
	V	Display code substitution	--
	=	Numeric substitution	--
	,	Comma (field separator)	+

Overlays

See 9.5	Call OVERLAY (fname, i, j, recall, k)	--

12. Example Program

12.1 CNOISE Model Before Application of Guidelines

```
      PROGRAM CNOISE(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,
     *                TAPE1,TAPE4,TAPE36)
C
C         THIS PROGRAM ANALYTICALLY CALCULATES SHIPPING NOISE.
C         AUTHOR: J. J. CORNYN (NORDA 321) WITH B. W. SCAIFE (ODSI)
C
C
      COMMON /PARTSM/ ALPHA,C,SINT,SRC
C
      DIMENSION FNAME(8),PHI0(72),PHI1(72),R(1000),TL(1000),TLDUM(1000)
     *          TOT(72),TOTDB(72),TOTINT(72)
C
      LOGICAL ENDRUN,FIELD,PRINTF
C
      DATA IUNIT1,IUNIT4,IUNT36,SIARM/1,4,36,4HSIAM/
C
      CALL DATE(IDATE)
      REWIND IUNT36
      WRITE (IUNT36,5000)  SIARM
      REWIND IUNIT1
      REWIND IUNIT4
      ENDRUN = .FALSE.
   10 CONTINUE
      READ(5,5000) FNAME
      IF(EOF(5) .NE. 0.0)  GO TO 8600
      ENDRUN = .TRUE.
      WRITE(6,6000) FNAME
      WRITE (IUNT36,5000)  FNAME
      READ (IUNIT4)  NOSEC,FNAME
      IF(EOF(IUNIT4) .NE. 0.0)  GO TO 8400
      WRITE (IUNT36,7000)  IDATE,NOSEC
      READ (IUNIT4)  (PHI0(J),PHI1(J),J=1,NOSEC)
      WRITE(6,6050) FNAME,NOSEC,(J,PHI0(J),PHI1(J),J=1,NOSEC)
      READ(5,5100) SRC,PRINTF
      WRITE(6,6070) SRC
      READ (IUNIT1)  FNAME,NOSECT,RDEPTH,FREQ,RINC,FIELD
      IF(EOF(IUNIT1) .NE. 0.0)  GO TO 8200
      WRITE(6,6100) FNAME,RDEPTH,FREQ,RINC
      IF(NOSECT.NE.NOSEC)  GO TO 8000
      IF(.NOT.FIELD)  GO TO 200
      WRITE(6,6150)
      READ (IUNIT1)  NTL,RMAXTL,SHDPTH,(TL(L),L=1,NTL)
      NTL1 = NTL-1
      WRITE(6,6200) NTL1,SHDPTH,RMAXTL,RINC,(TL(L),L=2,NTL)
      DO 100 L=1,NTL
       R(L) = (L-1)*RINC
  100 CONTINUE
C
      READ (IUNIT1)  NTLDUM,TLDUM(1),TLDUM(2),(TLDUM(L),L=1,NTLDUM)
  200 CONTINUE
      C = 1.0/ALOG10(EXP(1.0))
      OMNI = 0.0
      DO 3000 J=1,NOSEC
       READ (IUNIT4)  TOT(J),JSR
       IF(PRINTF)  WRITE(6,6250) J,PHI0(J),PHI1(J),TOT(J),JSR
       IF(FIELD)  GO TO 400
       READ (IUNIT1)  NTL,RMAXTL,SHDPTH,(TL(L),L=1,NTL)
       NTL1 = NTL-1
       IF(PRINTF)  WRITE(6,6200) NTL1,SHDPTH,RMAXTL,RINC,(TL(L),L=2,NTL)
       DO 300 L=1,NTL
        R(L) = (L-1)*RINC
  300  CONTINUE
C
       READ (IUNIT1)  NTLDUM,TLDUM(1),TLDUM(2),(TLDUM(L),L=1,NTLDUM)
  400  CONTINUE
       IF(PRINTF)  WRITE(6,6300)
       TOTINT(J) = 0.0
       DO 2000 I=1,JSR
        SINT = 0.0
        READ (IUNIT4)  RMIN,RMAX,SHIPS
        IF(RMIN.GE.RMAXTL)  GO TO 1700
        ALPHA = SHIPS/(RMAX*RMAX-RMIN*RMIN)*2.0
        K1 = RMIN/RINC+2.0
        TL1 = (TL(K1)-TL(K1-1))/RINC*(RMIN-R(K1-1))+TL(K1-1)
        IF(RMAX.GT.R(K1))  GO TO 500
        TL2 = (TL(K1)-TL(K1-1))/RINC*(RMAX-R(K1-1))+TL(K1-1)
        CALL PARSUM(RMAX-RMIN,RMIN,TL1,RMAX,TL2)
        GO TO 1500
  500   CONTINUE
        CALL PARSUM(R(K1)-RMIN,RMIN,TL1,R(K1),TL(K1))
        IF(RMAX.LT.RMAXTL)  GO TO 600
        K2 = NTL
        GO TO 900
  600   CONTINUE
        K2 = RMAX/RINC+1.0
        IF(RMAX.EQ.R(K2))  GO TO 900
        TL2 = (TL(K2+1)-TL(K2))/RINC*(RMAX-R(K2))+TL(K2)
        CALL PARSUM(RMAX-R(K2),R(K2),TL(K2),RMAX,TL2)
  900   CONTINUE
        K2 = K2-1
        IF(K1.GT.K2)  GO TO 1500
        DO 1000 L=K1,K2
         CALL PARSUM(RINC,R(L),TL(L),R(L+1),TL(L+1))
 1000   CONTINUE

 1500   CONTINUE
        IF(SINT .LE. 0.0001)  GO TO 1700
        SDB = 10.0*ALOG10(SINT)
        GO TO 1800
 1700   CONTINUE
        SDB = -40.0
 1800   CONTINUE
C
C           SECTOR-RANGE BIN OUTPUT
C
        IF(PRINTF)  WRITE(6,6350) I,RMIN,RMAX,SHIPS,SDB,SINT
        TOTINT(J) = TOTINT(J)+SINT
 2000  CONTINUE
C
       IF(TOTINT(J) .GT. 0.0001)  GO TO 2400
       TOTDB(J) = -40.0
       GO TO 2600
 2400  CONTINUE
       TOTDB(J) = 10.0*ALOG10(TOTINT(J))
 2600  CONTINUE
C
C           SECTOR OUTPUT
C
       IF(PRINTF)  WRITE(6,6400) J,TOTDB(J),TOTINT(J)
       WRITE (IUNT36,7100)  PHI0(J),PHI1(J),TOTDB(J)
       OMNI = OMNI+TOTINT(J)
 3000 CONTINUE
C
      WRITE(6,6450)  (J,PHI0(J),PHI1(J),TOT(J),TOTDB(J),TOTINT(J),
     *                J=1,NOSEC)
      IF(OMNI .GT. 0.0001)  GO TO 3400
      OMNIDB = -40.0
      GO TO 3600
 3400 CONTINUE
      OMNIDB = 10.0*ALOG10(OMNI)
 3600 CONTINUE
      WRITE(6,6500)  OMNIDB,OMNI
      GO TO 10
C
C
C*************************************************************************
C           INPUT
C*************************************************************************
C
 5000 FORMAT(8A10)
 5100 FORMAT(F10.0,9XL1)
C
C*************************************************************************
C           OUTPUT
C*************************************************************************
C
 6000 FORMAT(1H1,10X*SHIP NOISE RUN "*,8A10,*"*////)
 6050 FORMAT(1H0,*THE SHIP COUNT FILE IS IDENTIFIED BY "*,8A10,*"*///
     15X*THERE ARE*,I3,* SECTORS*//
     226X*SECTOR NUMBER  PHI0  TO  PHI1 (DEGREES)*/
     3(31XI3,3X2F10.1))
 6070 FORMAT(1H0,*SHIP SOURCE LEVEL IS*,F7.2,* DB.*)
 6100 FORMAT(1H0///* THE TRANSMISSION LOSS FILE IS IDENTIFIED BY "*,
     18A10,*"*//
     2* THERE IS A RECEIVER AT*,F11.2,* FT., A FREQUENCY OF*,F11.2,* HZ.
     3, AND A RANGE INCREMENT OF*,F8.2,* NAUTICAL MILES.*)
 6150 FORMAT(1H0,*THE TRANSMISSION LOSS DATA IS FOR A UNIFORM FIELD.*)
 6200 FORMAT(1H0,16X*THERE ARE*,I5,* TRANSMISSION LOSS VALUES FROM A SHI
     1PPING SOURCE DEPTH OF*,F8.2,* FT., OUT TO*,F8.2,* NAUTICAL MILES.*
     2/
     317X*THE FIRST VALUE IS AT RANGE = RANGE INCREMENT (*,F8.2,* N. M.)
     4, THEN PROCEED ALONG EACH LINE*//
     5(5X16F8.1))
 6250 FORMAT(1H1,*SECTOR NUMBER*,I3,* (*,F6.1,* TO*,F6.1,* DEGREES) HAS
     *A TOTAL OF*,F9.2,* SHIPS IN*,I3,* SECTOR-RANGE BINS.*)
 6300 FORMAT(1H0)
 6350 FORMAT(17X*SECTOR-RANGE BIN*,I3,* (*,F8.2,* TO*,F9.2,* N. M.) HAS*
     *,F9.2,* SHIPS, NOISE IS*,F10.4,* DB WITH INTENSITY =*,E11.4)
 6400 FORMAT(1H0,*FOR SECTOR*,I3,*, SECTOR NOISE IS*,F10.4,* DB WITH INT
     *ENSITY =*,E11.4///)
 6450 FORMAT(1H1,52X*SECTOR INFORMATION*/
     1               53X*------------------*//
     226X*SECTOR NUMBER  PHI0  TO  PHI1 (DEGREES)   SHIPS   NOISE (DB)
     3INTENSITY*//
     4(31XI3,3X2F10.1,F16.2,F11.4,E13.4))
 6500 FORMAT(1H1,21(/),32X73(1H*)/32X1H*,71X1H*/
     132X30H* THE OMNIDIRECTIONAL NOISE IS,F10.4,20H DB WITH INTENSITY =
     2E11.4,2H */32X1H*,71X1H*/32X73(1H*))
C
 7000 FORMAT(A10,5XI5)
 7100 FORMAT(3F10.2)
C
C
C*************************************************************************
C           ERRORS
C*************************************************************************
C
 8000 CONTINUE
      WRITE(6,8100)
 8100 FORMAT(1H0,*THE TRANSMISSION LOSS FILE, IUNIT1, HAS DATA THAT DOES
     1 NOT COVER THE SAME NUMBER OF SECTORS DEFINED FOR THIS SHIP NOISE
     2RUN.*)
       GO TO 9000
C
 8200 CONTINUE
      WRITE(6,8300)
 8300 FORMAT(1H0,*THE TRANSMISSION LOSS FILE, IUNIT1, HAS NO DATA FOR TH
     *IS RUN.*)
       GO TO 9000
C
 8400 CONTINUE
      WRITE(6,8500)
 8500 FORMAT(1H0,*THE GEOMETRY AND SHIP COUNT FILE, IUNIT4, HAS NO DATA
     *FOR THIS RUN.*)
       GO TO 9000
C
 8600 CONTINUE
      IF(ENDRUN) STOP
      WRITE(6,8700)
 8700 FORMAT(1H0,*THERE IS NO TITLE CARD FOR THIS RUN.*)
 9000 CONTINUE
      WRITE(6,9100)
 9100 FORMAT(//,10X,* --- PROGRAM CNOISE ABORTED IN MAIN PROGRAM ---*)
      CALL ABORT
      STOP 77777
      END
      SUBROUTINE PARSUM(DELR,R1,TL1,R
C
C
      COMMON /PARTSM/ ALPHA,C,SINT,SRC
C
      E1 = 0.1*(SRC-TL1)
      B = TL2-TL1


C
C           CHECK FOR FLAT TL CURVE SEGMENT
C
      IF(ABS(B) .LT. 0.001)  GO TO 100
      B = (-10.0)/C*DELR/B
      SINT = SINT+ALPHA*B*((R2-B)*EXP((SRC-TL2)/10.*C)-(R1-B)*EXP(C*E1))
      RETURN
C
C
  100 CONTINUE
```

NOT REPRODUCIBLE

```
C
C          EVALUATE INTEGRAL FOR SPECIAL CASE WHEN T(I)=T(I+1)
C
      SINT = SINT+ALPHA*10.**(S/10.)/2.*(R2*R2-R1*R1)
      RETURN
      END
```

12.2 CNOISE Model After Application of Guidelines

```
      PROGRAM CNOISE(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,
     1               TAPE1,TAPE4,TAPE36)
C
C   **   PURPOSE - THIS PROGRAM ANALYTICALLY CALCULATES THE HORIZONTALLY
C   **             DIRECTIONAL AMBIENT SEA NOISE DUE TO SHIPPING
C   **
C   **   AUTHOR -  JOHN J. CORNYN
C   **             NAVAL OCEAN RESEARCH AND DEVELOPMENT ACTIVITY
C   **             CODE 321
C   **             NSTL STATION, MS.  39529
C   **             (601)-688-4835
C   **
C   **        BASED ON EARLIER VERSION 1.0 BY AUTHOR AND
C   **
C   **             BARRY W. SCAIFE
C   **             OCEAN DATA SYSTEMS INC.
C   **             6000 EXECUTIVE BLVD SUITE 615
C   **             ROCKVILLE, MD  20852
C   **             (301)-881-3031
C   **
C   **   VERSION 2.0
C   **   DATE OF FIRST COMPILATION -APRIL, 1979
C   **   DATE PROGRAM LAST UPDATED - 4 NOV 1980
C   **     PROGRAM WRITTEN FOR NORDA CODE 321
C   **                        SPONSOR-SEAS PROJECT, NORDA CODE 520
C   **
C   **   PROCESSING PERFORMED BY PROGRAM -THE INPUTS TO CNOISE ARE
C   **    ********************************
C   **     SECTOR GEOMETRY,TRANSMISSION LOSS CURVES FOR EACH SECTOR,
C   **     SHIP COUNTS IN RANGE SECTOR BINS, AND SHIP SOURCE LEVELS.
C   **     THE OUTPUTS GENERATED ARE MEAN DIRECTIONAL AND
C   **     OMNIDIRECTIONAL NOISE LEVELS. IF WE ASSUME A FLAT EARTH,
C   **     THE AREA, A, OF A RANGE SECTOR BIN IS GIVEN BY THE EXPRESSION
C   **     A=0.5*(R2**2-R1**2)*(THETA2-THETA1).
C   **     IF THIS RANGE SECTOR BIN HAS N UNIFORMLY DISTRIBUTED SHIPS,
C   **     THEN THE DENSITY OF SHIPS PER UNIT RANGE ,LAMBDA(R), IN
C   **     THIS RANGE SECTOR BIN CAN BE OBTAINED BY  INTEGRATING THE
C   **     SHIP DENSITY PER UNIT AREA OVER THETA, SO
C   **      LAMBDA(R)= 2*N*R/(R2**2-R1**2).
C   **     IF WE ASSUME THE BEGINNING OF RANGE BIN J OCCURS AT POINT
C   **     K(J) ON THE TRANSMISSION LOSS CURVE, AND ENDS AT POINT
C   **     M(J), AND IF WE ASSUME THE TRANSMISSION LOSS CURVE CAN BE
C   **     REPRESENTED AS A LINEAR FUNCTION OF RANGE OVER THE BIN,
C   **     THEN THE INTENSITY AT THE RECEIVER LOCATION BETWEEN
C   **     AZIMUTHAL ANGLES THETA1 AND THETA2 DUE TO N SUCH RANGE-SECTOR
C   **     BINS IS GIVEN BY
C   **        I= SUM(FOR J=1 TO N OF(SUM FOR I=K(J)TO M(J) OF F(I,J)))
C   **     WHERE
C   **      F(I,J)=INTEGRAL OVER R FROM R(I) TO R(I+1) OF
C   **               LAMBDA(R)*10**((S-T(I,R))/10)
C   **      WHERE  S IS THE SHIP SOURCE LEVEL (IN DB)
C   **             T(I,R) = THE TRANSMISSION LOSS IN SECTOR I AT
C   **                      RANGE R
C   **     THIS INTEGRAL CAN BE EVALUATED EXACTLY.
C   **     IF T(I,R)=A(I)+B(I)*R FOR I=K(J) TO M(J)
C   **     THEN IF B(I).NE. 0
C   **       F(I,J)=ALPHA(J)*EXP(F(I))*
C   **                 EXP(D(I)*R(I+1))*(D(I)*R(I+1)-1)/D(I)**2 -
C   **                 EXP(D(I)*R(I))*(D(I)*R(I)-1)/D(I)**2)
C   **
C   **     WHERE  F(I)=C*(S-A(I))/10 AND
C   **            D(I)=-C*B(I)/10  AND
C   **            C=1./LN(EXP(1))
C   **            LN=NATURAL LOG FUNCTION
C   **            EXP=EXPONENTIAL FUNCTION
C   **     IF B(I)=0 THEN
C   **       F(I,J)=ALPHA(J)*10**((S-A(I))/10)*(R(I+1)**2-R(I)**2)/2
C   **     WHERE ALPHA(J)=LAMBDA(R)/R FOR THE J-TH RANGE SECTOR BIN
C   **
C   **     IN SUMMARY, CNOISE DETERMINES THE F(I,J), SUMS THEM,
C   **     AND TAKES THEIR LOG TO BASE 10 TO OBTAIN THE SECTOR
C   **     AND OMNIDIRECTIONAL NOISE LEVELS DUE TO SURFACE SHIPS.
C   **
C   **     THE CNOISE SYSTEM IS COMPRISED OF THREE, OPTIONALLY FOUR,
C   **     PROGRAMS: SIANPR,SIANTL,CNOISE, AND OPTIONALLY,SIANAD.
C   **     PROGRAMS SIANPR,SIANTL,AND SIANAD WERE ORIGINALLY DESIGNED
C   **     TO WORK WITH THE MONTE CARLO AMBIENT NOISE AND SIGNAL EXCESS
C   **     MODEL SIAN.  THE SIANAD PROGRAM WAS DEVELOPED AS A SIAN
C   **     POST-PROCESSOR.
C   **
C   **     BRIEFLY, PROGRAM SIANPR GENERATES SHIP COUNTS IN SPECIFIED
C   **     SECTOR-RANGE BINS.  SIANPR UTILIZES A PACKED SHIPPING DENSITY
C   **     FILE AND CARD INPUT THAT DEFINES A SECTOR-RANGE BIN GEOMETRY.
C   **     A SPECIALLY FORMATTED TRANSMISSION LOSS(TL) FILE IS
C   **     GENERATED BY SIANTL.  TL CURVES ARE INPUT TO SIANTL FROM
C   **     CARDS, A 'PE' FORMATTED FILE, OR A 'FACT' FORMATTED FILE.
C   **     THEN, FROM SELECTED TL INPUT FILES FOR EACH SECTOR- ONE
C   **     FOR SHIPPING AND ONE FOR A TARGET SOURCE - SIANTL GENERATES
C   **     TWO TL CURVES HAVING A CONSTANT RANGE INCREMENT. CNOISE
C   **     UTILIZES THE SHIP COUNT AND GEOMETRY FILE, THE TL FILE,
C   **     AND CARD INPUTS TO GENERATE FOR EACH SECTOR THE NOISE DUE
C   **     TO SHIPPING.  THE SHIPPING NOISE AND THE SECTOR ANGLES
C   **     ARE OUTPUT TO A FILE IN THE SAME FORMAT AS A SIAN GENERATED
C   **     FILE.  SIANAD IS A PROGRAM THAT INTENSITY ADDS THE SIAN
C   **     (OR CNOISE) SHIPPING NOISE WITH FANMI GENERATED WIND NOISE.
C   **
C   **   REFERENCES-  J.J.CORNYN,"A SIMPLE ANALYTICAL AMBIENT
C   **   **********   NOISE MODEL," NORDA CODE 321, TEXT OF
C   **                PAPER PRESENTED AT 99TH MEETING OF
C   **                ACOUSTICAL SOCIETY OF AMERICA, ATLANTA, GEORGIA,
C   **                21-25 APRIL 1980
C   **
C   **                B.W. SCAIFE,"CNOISE COMPUTER PROGRAM  SYSTEM,"
C   **                MEMO TO J.CORNYN, 6 OCTOBER 1979,ODSI,INC.
C   **
C   **   *****************************************************************
C   **   CARD INPUT
C   **   **********
C   **
C   **   CARD  FORMAT   COLUMN NO.        VARIABLE NAME
C   **   ----  ------   ----------        -------------
C   **
C   **    1    20A4       1-80            (FNAME(I),I=1,20)
C   **    2    F10.0      1-10*           SRC
C   **    3    L1          80             PRINTF
C   **    (* MAY BE POSITIONED ANYWHERE IN SPECIFIED COLUMNS WITH
C   **       DECIMAL POINT EXPRESSED)
C   **
C   **   VARIABLE  NAME                   DESCRIPTION
C   **   --------------                   -----------
C   **   (FNAME(I),I=1,20        AN EIGHTY CHARACTER HEADER FOR THE
C   **                           OUTPUT FILE
C   **     SRC                   SHIP SOURCE LEVEL (IN DB)
C   **     PRINTF                PRINT OUTPUT FLAG
C   **                            F  PRINT A SUMMARY
C   **                            T  PRINT A SUMMARY AND DETAILED
C   **                               SECTOR-RANGE BIN DATA AND
C   **                               TRANSMISSION LOSS PROFILES
C   **                           (IF PRINTF IS LEFT BLANK ,"F" IS ASSUMED.
C   **
C   ** *****************************************************************
C   **
C   **
C   ** FILE DESCRIPTIONS
C   ** *****************
C   **   UNIT NO.  UNIT NAME  TYPE      CONTENTS
C   **   --------  ---------  ----      --------
C   **     5        ICARD     INPUT     CARD INPUT
C   **     6        IPRINT    OUTPUT    PRINTER OUTPUT
C   **     1        ITL       INPUT     TRANSMISSION LOSS CURVES OUTPUT
C   **                                  BY SIANTL PROGRAM
C   **
C   **     4        ISHIPS    INPUT     SECTOR-RANGE BIN GEOMETRY
C   **                                  AND SHIP COUNT FILE
C   **
C   **    36        INOISE    OUTPUT    DATE,NUMBER OF SECTORS,
C   **                                  AZIMUTHAL ANGES DEFING SECTORS,
C   **                                  NOISE LEVEL IN SECTORS(DB)
C   **
C   **
C   **
C   **   ALL FILES ARE WRITTEN IN THE MAIN PROGRAM (CNOISE)
C   **
C   **   THE PRINTED OUTPUT INCLUDES:
C   **     TITLE CARD,NUMBER OF SECTORS AZIMUTHAL ANGLES DEFINING
C   **     SECTORS,INPUT SHIP SOURCE LEVEL, TRANSMISSION LOSS
C   **     CURVES FOR EACH SECTOR, FOR EACH RANGE-SECTOR-BIN
C   **     IN A SECTOR THE NUMBER OF SHIPS,NOISE DUE TO THOSE SHIPS
C   **     IN DB,INTENSITY DUE TO THOSE SHIPS), THE TOTAL NOISE(IN DB)
C   **     FOR THE SECTOR, TOTAL NOISE INTENSITY FOR SECTOR,
C   **     THE TOTAL OMNIDIRECTIONAL NOISE (IN DB AND BY INTENSITY)
C   **
C   **
C   *******************************************************************
C   ****  OPTIONS AVAILABLE ...  SEE CARD INPUT
C   ****  *****************
C   **
C   *******************************************************************
C   **
      LOGICAL  ENDRUN,FIELD,PRINTF
      DIMENSION FNAME(20),PHI1(72),PHI2(72),R(1000),TL(1000),
     1          TLDUM(1000),NTOT(72),TOTDB(72),TOTINT(72)
C
C   **     FNAME          TITLE CARD
C   **     PHI1           INITIAL AZIMUTHAL ANGLE (DEG) FOR SECTOR J
C   **     PHI2           FINAL AZIMUTHAL ANGLE (DEG) FOR SECTOR J
C   **     R(I)           RANGE (NM) OF ITH POINT OF TL CURVE
C   **     TL(I)          TRANSMISSION LOSS (DB) AT RANGE R(I)
C   **     TLDUM          TEMP ARRAY FOR HOLDING TRANSMISSION LOSS VALUES
C   **     NTOT           TOTAL NUMBER OF SHIPS IN JTH SECTOR
C   **     TOTDB          TOTAL DB LEVEL FOR JTH SECTOR (IN DB)
C   **     TOTINT         TOTAL INTENSITY FOR JTH SECTOR
C   **     ENDRUN         FLAG WHICH IS SET TO T WHEN THERE ARE NO MORE
C   **                    CASES TO BE RUN
C   **     FIELD          FLAG WHICH IS READ FROM THE TL FILE, SET TO TRUE
C   **                    IF THIS IS AN AZIMUTHALLY UNIFORM FIELD.
C   **     PRINTF         FLAG  SEE CARD INPUT DESCRIPTION ABOVE.
C
C
      COMMON/PARTSA    ALPHA,C,SRC
C
C   **      ALPHA        SHIP DENSITY PER UNIT RANGE DIVIDED BY RANGE
C   **      C            A CONSTANT, =1./ALOG10(EXP(1.0))
C   **      SRC          SHIP SOURCE LEVEL (IN DB)
C   **
C   **   DEFINE LOGICAL UNIT NUMBERS AND HEADER  INFO
C
      DATA ICARD,ITL,ISHIPS,INOISE,IPRINT, SIANH /5,1,4,36,6,4HSIAN /
C
C   **     DETERMINE DATE FROM OPERATING SYSTEM
C          DATA IS A SYSTEM DEPENDENT ROUTINE THAT RETURNS THE DATE
C          CHARACTERS IN CHARACTER FORMAT IN THE VARIABLE IDATE
C
      CALL DATE(IDATE)
C
C
C   **     REWIND THE OUTPUT NOISE FILE
C
      REWIND INOISE
C
C   **     WRITE THE WORD 'SIAN' ON THE NOISE FILE
C
      WRITE(INOISE,9000) SIANH
C
C   **     REWIND THE SHIPPING AND TRANSMISSION LOSS FILES
C
      REWIND ITL
      REWIND ISHIPS
C
C   **     THE FLAG SIGNIFYING END OF CARD INPUT IS SET TO FALSE
C
      ENDRUN = .FALSE.
C
 10   CONTINUE
C
C   **     READ IN TITLE CARD
```

NOT REPRODUCIBLE

```
C
      READ(ICARD,9000) (FNAME(I),I=1,20)
C
C   **    CHECK TO SEE IF AN END-OF-FILE CARD HAS BEEN ENCOUNTERED
C         IF IT HAS GO TO STATEMENT 20
C
      IF(EOF(ICARD) .EQ. 0.0) GO TO 20
C
C   **
C
      IF(.NOT.ENDRUN) GO TO 5000
C
C   **   STOP PROGRAM NORMALLY
C
      WRITE(IPRINT,9200)
      STOP
C
 5000 WRITE(IPRINT,9870)
C
C   **    ABORT PROGRAM
C
      WRITE(IPRINT,9100)
      STOP 7777
C
 20   CONTINUE
      ENDRUN = .TRUE.
C
C   **    WRITE TITLE   ON PRINTER AND ON NOISE FILE
C
      WRITE(IPRINT,9600) (FNAME(I),I=1,20)
      WRITE(INOISE,9000) (FNAME(I),I=1,20)
C
C   **    READ NUMBER OF SECTORS AND TITLE FROM TL FILE
C
      READ(ITL)  NOSEC,(FNAME(I),I=1,20)
C
C   ** CHECK FOR AN END OF FILE
C
      IF( EOF(ITL) .EQ. 0.0) GO TO 30
C
C
C   **    END OF FILE ENCOUNTERED  ON  TL FILE -WRITE MESSAGE AND ABORT
C
      WRITE(IPRINT,9850)
      WRITE(IPRINT,9100)
      STOP 7777
C
 30   CONTINUE
C
C   **    PUT DATE AND NUMBER OF SECTORS ON NOISE FILE
C
      WRITE(INOISE,9700) IDATE,NOSEC
C
C   **    READ SECTOR ANGLES
C
      READ(ITL) (PHI1(J),PHI2(J), J=1,NOSEC)
C
C   **    PRINT OUT  TITLE CARD, NUMBER OF SECTORS,SECTOR NUMBERS,ANGLES
C
      WRITE(IPRINT,9605) (FNAME(I),I=1,20),NOSEC,(J,PHI1(J),PHI2(J),
     1   J=1,NOSEC)
C
C   **    READ IN  SHIP SOURCE LEVEL AND PRINTOUT FLAG
C
      READ(ICARD,9510) SRC, PRINTF
C
C   **    PRINT OUT SOURCE LEVEL OF SHIP
C
      WRITE(IPRINT,9607) SRC
C
C   **    READ TITLE, NUMBER OF SECTORS,RECEIVER DEPTH,FREQUENCY,
C   **          RANGE INCREMENT, AND FIELD FLAG FROM THE TL CURVE FILE
C
      READ(ITL)(FNAME(I),I=1,20),NOSECT,RDEPTH,FREQ,RINC,FIELD
C
C   **    CHECK FOR AN END OF FILE
C
      IF (EOF(ITL) .EQ. 0.0)  GO TO 40
C
C   ** END-OF-FILE ENCOUNTERED ON TL FILE -WRITE MESSAGE AND ABORT
C
      WRITE(IPRINT,9830)
      WRITE(IPRINT,9100)
      STOP 7777
C
 40   CONTINUE
C
C   **    PRINT OUT TITLE ON  TL FILE, RECEIVER DEPTH, FREQUENCY,
C   **    AND RANGE INCREMENT
C
      W..TE(IPRINT,9610) (FNAME(I),I=1,20),RDEPTH,FREQ, RINC
C
C   **    CHECK TO SEE  IF NUMBER OF SECTORS ON  TL  AND SHIPPING
C   **    FILES AGREE
C
      IF(NOSECT .EQ. NOSEC)  GO TO 50
C
C   **   DONT AGREE  -WRITE ERROR MESSAGE AND ABORT
C
      WRITE(IPRINT,9810)
      WRITE(IPRINT,9100)
      STOP 7777
C
 50   CONTINUE
C
C   **    CHECK TO SEE IF THE  TL FIELD IS AZIMUTHALLY UNIFORM
C
      IF( .NOT. FIELD) GO TO 200
C
C   **   UNIFORM FIELD ,SO WRITE OUT MESSAGE
C
      WRITE(IPRINT,9615)
C
C   **    READ NUMBER OF TL POINTS,MAXIMUM  RANGE OF TL CURVE IN NM,
C   **    SHIP SOURCE DEPTH(IN FEET),   TRANSMISSION LOSS VALUES
C
      READ(ITL) NTL,RMAXTL, SHDPTH,(TL(L),L=1,NTL)
      NTL1=NTL-1
C
C   **    PRINT OUT  TL CURVE
C
      WRITE(IPRINT,9620)NTL1,SHDPTH,RMAXTL,RINC,(TL(L),L=2,NTL)
C
C   ** COMPUTE RANGES OF TL POINTS,ASSUMING EQUALLY SPACED
C
      DO 100 L =1,NTL
         R(L)=(L-1)*RINC
 100  CONTINUE
C
C   **READ IN SOME DUMMY TL VALUES FROM THE TL FILE
C
C
      READ(ITL) NTLDUM,TLDUM(1),TLDUM(2),(TLDUM(L),L=1,NTLDUM)
C
C
 200  CONTINUE
C
C   **    CALCULATE A CONSTANT FOR INTEGRAL
C
        C=1.0/ALOG10(EXP(1.0))
C
C   **    ZERO OUT THE OMNIDIRECTIONAL INTENSITY LEVEL
C
      OMNI=0.0
C
C   **    LOOP OVER SECTORS
C
      DO 3000 J=1,NOSEC
C
C   **    READ NUMBER OF SHIPS IN JTH SECTOR, AND NUMBER OF SECTOR RANGE
C   **      BINS
C
      READ(ISHIPS) TOT(J),JSR
C
C   **    PRINT OUT THE SECTOR NUMBER, SECTOR ANGLES, NO. OF SHIPS,
C   **      NO. OF BINS
C
        IF(PRINTF) WRITE(IPRINT,9625) J,PHI1(J),PHI2(J),TOT(J),JSR
C
C   **    CHECK TO SEE IF THERE IS A UNIFORM  TL  FIELD
C
        IF(FIELD) GO TO 400
C
C   **    NOT UNIFORM, SO READ IN   TL CURVE FOR THE SECTOR
C
        READ(ITL) NTL,RMAXTL,SHDPTH,(TL(L),L=1,NTL)
        NTL1=NTL-1
C
C   **    PRINTOUT TL INFO IF REQUESTED
C
        IF(PRINTF) WRITE(IPRINT,9620) NTL1,SHDPTH,RMAXTL,RINC,
     1            (TL(L),L=2,NTL)
C
C   **    COMPUTE RANGES FOR THIS TL CURVE
C
        DO 300  L=1,NTL
           R(L)=(L-1)*RINC
 300    CONTINUE
C
C   **    READ IN DUMMY VALUES OF TL CURVES
C
        READ(ITL) NTLDUM,TLDUM(1),TLDUM(2),(TLDUM(L),L=1,NTLDUM)
C
 400    CONTINUE
C
C   **    IF PRINTOUT REQUESTED,GO TO A NEW PAGE
C
         IF (PRINTF) WRITE(IPRINT,9630)
C
C   **    ZERO TOTAL INTENSITY FOR JTH SECTOR
C
         TOTINT(J)=0.0
C
C   **    LOOP OVER RANGE BINS FOR THE JTH SECTOR
C
         DO 2000 I=1 ,JSR
C
C   **    INITIALIZE  INTENSITY FOR THIS RANGE BIN
C
            SINT =0.0
C
C   **    READ MINIMUM RANGE OF BIN (RMIN),MAXIMUM RANGE OF BIN (RMAX),
C   **         AND NUMBER OF SHIPS IN BIN (SHIPS)
C
            READ(ISHIPS) RMIN,RMAX,SHIPS
C
C   **    CHECK TO SEE IF MINIMUM RANGE OF BIN EXCEEDS MAXIMUM
C   **    RANGE OF TL CURVE
C
            IF( RMIN .GE. RMAXTL) GO TO 1700
C
C   **    IT DOESN'T,SO COMPUTE THE SHIP DENSITY PER UNIT RANGE
C   **    DIVIDED BY RANGE,I.E. ALPHA, FOR  THIS BIN
C
            ALPHA =(SHIPS/(RMAX*RMAX-RMIN*RMIN))*2.0
C   **
C   **    GET APPROPRIATE INDEX IN TL CURVE CORRESPONDING TO
C   **    MINIMUM RANGE OF BIN AND COMPUTE TL AT RMIN USING LINEAR
C   **    INTERPOLATION
C
            K1=(RMIN/RINC)+2.0
            TL1=((TL(K1)-TL(K1-1))/RINC)*(RMIN-R(K1-1))+TL(K1-1)
C
C   **    CHECK TO SEE IF RANGE-SECTOR BIN IS SMALLER THAN RANGE
C   **    INCREMENT ON TL CURVE
C
            IF(RMAX .GT. R(K1)) GO TO 500
C
C   **    IT IS,SO INTERPOLATE A TL VALUE AT RMAX
C
            TL2=((TL(K1)-TL(K1-1))/RINC)*(RMAX-R(K1-1)) + TL(K1-1)
C
C   **    CALL SUBROUTINE PARSUM  TO EVALUATE THE INTEGRAL OVER THE
C   **    INTERVAL  RMAX TO RMIN
C
            CALL PARSUM(RMAX-RMIN,RMIN,TL1,RMAX,TL2,SINT)
C
            GO TO 1500
C
 500        CONTINUE
C
C   **    EVALUATE THE CONTRIBUTION TO INTEGRAL FROM THE
C   **    MINIMUM RANGE OF BIN TO THE RANGE R(K1)
C
            CALL PARSUM(R(K1)-RMIN,RMIN,TL1,R(K1),TL(K1),SINT)
C
C   **    CHECK TO SEE IF MAXIMUM RANGE OF BIN IS LESS THEN THE MAXIMUM
C   **    RANGE OF THE TL CURVE
C
            IF(RMAX .LT. RMAXTL ) GO TO 600
```

NOT REPRODUCIBLE

```
C
C    **   IF ISN'T SO SET K2 EQUAL INDEX OF LAST VALUE OF TL CURVE
C
            K2 = NTL
            GO TO 900
C
 600        CONTINUE
C
C    **   IT IS LESS THAN RMAXTL, SO FIND THE INDEX CORRESPONDING
C    **   TO THE MAXIMUM RANGE (RMAX)
C
            K2 = (RMAX/RINC) + 1.0
C
C    **   CHECK TO SEE IF RMAX CORRESPONDS EXACTLY TO R(K2) TO SEE
C    **   IF INTERPOLATION IS NECESSARY
C
            IF( RMAX .EQ. R(K2) ) GO TO 900
C
C    **   EVALUATE CONTRIBUTION TO INTEGRAL FROM RANGE R(K2) TO RMAX
C
            TL2 = ((TL(K2+1)-TL(K2))/RINC)*(RMAX-R(K2)) + TL(K2)
            CALL PARSUM(RMAX-R(K2),R(K2),TL(K2),RMAX,TL2,SINT)
C
 900        CONTINUE
C
            K2= K2-1
C
C    **    CHECK TO SEE INDICES RUN CORRECTLY.
C
            IF( K1 .GT. K2) GO TO 1500
C
C    **  NOW EVALUATE CONTRIBUTION TO INTEGRAL FROM RANGE R(K1) TO
C    **  RANGE R(K2)
C
            DO 1000 L =K1,K2
               CALL PARSUM( RINC,R(L),TL(L),R(L+1),TL(L+1),SINT)
 1000       CONTINUE
C
 1500       CONTINUE
C
C    ** CHECK TO SEE OF THE TOTAL INTENSITY CONTRIBUTED TO THE RANGE-BIN
C    ** UP TO THIS POINT (SINT) IS .LE. 10**(-4) IF IT IS SET THE
C    ** DB VALUE FOR BIN TO -40DB
C
            IF(SINT .LE. 0.0001) GO TO 1700
C
C    ** CONVERT INTENSITY TO DB
C
            SDB=10.0 *ALOG10(SINT)
            GO TO 1800
C
 1700       CONTINUE
            SDB = -40.
C
 1800       CONTINUE
C
C    **     PRINT OUT INFO FOR SECTOR RANGE BIN,IF REQUESTED BY USER
C
            IF( PRINTF ) WRITE(IPRINT,9635) I,RMIN,RMAX,SHIPS,SDB,SINT
C
C    **
C    **      ADD IN INTENSITY OF THIS RANGE SECTOR BIN TO THAT OF
C    **      THE ENTIRE JTH SECTOR
C
             TOTINT(J) =TOTINT(J) + SINT
C
C    **   END OF LOOP ON  RANGE BINS  FOR  JTH  SECTOR
C
 2000      CONTINUE
C
C    **   CHECK TO SEE IF TOTAL INTENSITY FOR SECTOR IS GREATER THAN
C    **   -40DB. IF IT ISN'T ,SET IT TO -40DB.
C
          IF(TOTINT(J) .GT. 0.0001) GO TO 2400
          TOTDB(J) = -40.0
          GO TO 2600
 2400     CONTINUE
          TOTDB(J) = 10.0*ALOG10(TOTINT(J))
 2600     CONTINUE
C
C    **    WRITE OUT SUMMARY INFO FOR THE ENTIRE SECTOR, IF REQUESTED
C
          IF(PRINTF) WRITE(IPRINT,9640) J,TOTDB(J),TOTINT(J)
C
C    **    WRITE SECTOR ANGLES AND TOTAL SECTOR DB VALUE ON NOISE FILE
C
          WRITE(INOISE,9710) PHI1(J),PHI2(J),TOTDB(J)
C
C    **  ADD IN CONTRIBUTION OF THIS SECTOR TO TOTAL  OMNIDIRECTIONAL
C    **  INTENSITY( OMNI)
C
          OMNI = OMNI + TOTINT(J)
C
C    **  END  OF  LOOP ON SECTORS (J)
C
 3000     CONTINUE
C
C    **   PRINT OUT SUMMARY INFO FOR SECTORS
C
          WRITE(IPRINT,9645)(J,PHI1(J),PHI2(J),TOT(J),TOTDB(J),TOTINT(J),
     1                      J =1,NOSEC)
C
C    **  IF THE OMNIDIRECTIONAL INTENSITY IS LESS THAN -40DB, SET IT TO
C    **   -40 DB
C
          IF(OMNI .GT. 0.0001) GO TO 3400
          OMNIDB =  -40.0
          GO TO 3600
 3400     CONTINUE
C
C    **   CONVERT OMNI INTENSITY TO DB
C
          OMNIDB = 10.0* ALOG10( OMNI)
 3600     CONTINUE
C
C    **  WRITE OMNI DB VALUE AND OMNIDIRECTIONAL INTENSITY ON PRINTOUT
C
          WRITE(IPRINT,9650) OMNIDB,OMNI
C    **  RETURN TO READ NEXT SET OF CARD INPUT
C
          GO TO  10
C  ************************************************************************
C                          INPUT FORMATS
C  ************************************************************************
C
 9510 FORMAT(F10.4, 9X, L1)
C
C**************************************************************************
C                          INPUT/OUTPUT FORMATS
C**************************************************************************
C
 9000 FORMAT(20A4)
C
C**************************************************************************
C                          OUTPUT FORMATS
C**************************************************************************
C
 9200 FORMAT(//,10X,29HNORMAL TERMINATION OF CNOISE )
 9600 FORMAT(1H1, 19X, 16HSHIP NOISE RUN ', 20A4, 1H', ///)
 9605 FORMAT(1H0, 38HTHE SHIP COUNT FILE IS IDENTIFIED BY ', 20A4,
     1         1H', ///,
     2          5X,  9HTHERE ARE, I3, 9H SECTORS), //,
     3         26X, 39HSECTOR NUMBER  PHI1  TO PHI2 (DEGREES), /,
     4         31X, I3, 3X, 2F10.1)
 9607 FORMAT(1H0,  20HSHIP SOURCE LEVEL IS, F7.2, 4H DB. )
 9610 FORMAT(1H0, ///, 46H THE TRANSMISSION LOSS FILE IS IDENTIFIED BY '
     1         ,20A4, 1H', //,
     2         23H THERE IS A RECEIVER AT, F11.2, 20H FT., A FREQUENCY OF,
     3     F11.2,  30H HZ., AND A RANGE INCREMENT OF, F9.2,
     4         16H NAUTICAL MILES.)
 9615 FORMAT( 1H0, 47HTHE TRANSMISSION LOSS DATA FOR A UNIFORM FIELD.)
 9620 FORMAT( 1H0, 16X,  9HTHERE ARE, I5, 54HTRANSMISSION LOSS VALUES FR
    1OM SHIPPING SOURCE DEPTH OF, F8.2, 12H FT., OUT TO, F9.2,
    2 15HNAUTICAL MILES.,/
    3 17X, 47HTHE FIRST VALUE IS AT RANGE = RANGE INCREMENT (, F8.2,
    4  35HNM.), THEN PROCEED ALONG EACH LINE), //,
    5  (5X , 16F8.1 ))
 9625 FORMAT( 1H1, 13HSECTOR NUMBER, I3, 2H (, F6.1, 4H TO, F6.1,
    1  23H DEGREES HAS A TOTAL OF, F9.2,  9H SHIPS IN, I3,
    2  10H SECTOR-RANGE BINS
 9630 FORMAT(1H0 )
 9635 FORMAT( 17X , 18HSECTOR-RANGE BIN, I3,  2H (, F8.2, 3H TO, F9.2,
    1    8HNM.) HAS, F8.2 , 16H SHIPS, NOISE IS, F10.4,
    2  20H DB WITH INTENSITY =, E11.4)
 9640 FORMAT( 1H0, 10HFOR SECTOR, I3,  17H, SECTOR NOISE IS, F10.4,
    1 20H DB WITH INTENSITY = , E11.4, /// )
 9645 FORMAT( 1H1, 58X, 18HSECTOR INFORMATION,/,
    1               53X, 18H------------------,//,
    2      26X,  68HSECTOR NUMBER PHI1 TO PHI2 (DEGREES)    SHIPS   NOIS
    3E (DB) INTENSITY, //,
    4 31X, I3, 3X, 2F10.1, F16.2, F11.4, E13.4 )
 9650 FORMAT( 1H1, 21(1H ), 32X, 73(1H*), /,
    1          32X, 1H*, 71X, 1H*, /,
    2          32X, 30H* THE OMNIDIRECTIONAL NOISE IS, F10.4,
    3          20H DB WITH INTENSITY =, E11.4, 2H *, /,
    4          32X, 1H*, 71X, 1H*, /,
    5          32X, 73(1H*) )
 9700 FORMAT( A10, 5X, I5 )
 9710 FORMAT( 3F10.2 )
C
C    **********************************************************************
C                              ERROR  MESSAGES
C    **********************************************************************
C
 9810 FORMAT(1H0, 120HTHE TRANSMISSION LOSS FILE, ITL, HAS DATA THAT DOE
    1S NOT COVER THE SAME NUMBER OF SECTORS DEFINED FOR THIS SHIP NOISE
    2RUN. )
C
 9830 FORMAT(1H0,  58HTHE TRANSMISSION LOSS FILE, ITL, HAS NO DATA FOR T
    1HIS RUN. )
C
 9850 FORMAT(1H0,  67HTHE GEOMETRY AND SHIP COUNT FILE, ISHIPS, HAS NO D
    1ATA FOR THIS RUN. )
C
 9870 FORMAT(1H0,  36HTHERE IS NO TITLE CARD FOR THIS RUN.)
C
 9100 FORMAT(//, 10X,  46H --- PROGRAM CNOISE ABORTED IN MAIN PROGRAM---
    1 )
      END
      SUBROUTINE PARSUM(DELR,R1,TL1,R2,TL2,SINT)
C
C    ** PURPOSE -THIS SUBROUTINE EVALUATES THE INTEGRAL OF
C    **            LAMBDA(R)*10**((SRC-TL)/10) OVER A SINGLE
C    **            SEGMENT OF RANGE ,DELR, RUNNING FROM RANGE R1
C    **            TO RANGE R2. HERE LAMBDA(R) IS DEFINED IN
C    **            PROGRAM CNOISE.THE TRANSMISSION LOSS IS
C    **            ASSUMED TO BE LINEAR OVER DELR.
C
C    **   PARAMETER       TYPE            DESCRIPTION
C    **   ---------       ----            -----------
C    **     DELR           INPUT          -RANGE INTERAL (IN NM) OF WHICH
C    **                                    INTEGRAL IS EVALUATED
C    **      R1            INPUT          -INITIAL RANGE (NM) OF INTERVAL
C    **      TL1           INPUT          -TRANSMISSION LOSS(IN DB)AT R1
C    **      R2            INPUT          -FINAL   RANGE (NM) OF INTERVAL
C    **      TL2           INPUT          -TRANSMISSION LOSS(IN DB) AT R2
C    **      SRC           INPUT          -SHIP SOURCE LEVEL (IN DB)
C    **      SINT          OUTPUT         -VALUE OF INTEGRAL (INTENSITY)
C    **
C
      COMMON /PARTSM/ ALPHA,C,SRC
C
      E1 = 0.1*(SRC-TL1)
      B  = TL2-TL1
C
C    **    CHECK TO A FLAT TL CURVE SEGMENT
C
      IF (ABS(B) .LT. 0.001)  GO TO 100
      B =((-10.0)/C)*(DELR/B)
      SINT =SINT + ALPHA *B*((R2-B)*EXP((SRC-TL2)*0.1*C)
     1                        -(R1-B)*EXP(C*E1))
C
      GO TO 900
C
C
100   CONTINUE
C
C    **    EVALUATE INTEGRAL FOR SPECIAL CASE WHEN T(J)=T(J+1)
C
      SINT = SINT +ALPHA*((10.0**(E1))/2.)*(R2*R2-R1*R1)
C
900   RETURN
      END
*EOR
```

NOT REPRODUCIBLE

13. Conclusion

This document has addressed th c of programming style. Its ap ion will certainly require more work for the programmer in the short term, but in the long term, the rewards to the programmer, his organization, and other organizations and individuals using the programs could be substantial. The programmer should gain in the long term, because his programs will be easier for him to understand, and will be more likely to be used by others. The organization will gain because the costs it incurs for maintenance and software conversion will be lower. Other individuals and organizations will profit because less effort will be required to understand, use, extend, and adapt the programs.

14. References

Berkowitz, R. L. (1970). A Comparison of Some FORTRAN Languages, Naval Research Laboratory, Washington, D.C., October, NRL Memorandum Report 2191, NRL Computer Bulletin 21, NRL Problem 56R06-41, Project No. A-37-533-00/6521/ WF08-051-702, 34 p.

Control Data Corporation (1976). FORTRAN Extended Version 4 Reference Manual, Revised Edition, Cyber 70 Series. Revision K. Sunnyvale, Calif., Pub. No. 60305601.

Fleiss, J. E., G. W. Phillips, A. Edwards, L. Rieder (1974). Programming for Transferabil: y, International Computer Systems, Inc. (Unpublished).

Jacobs, G. (1976). Computer Programming Standards, Ocean Data Systems, Inc., Rockville, Maryland (Unpublished internal memo).

Jensen, R. W., C. C. Tonies (1979). Software Engineering, Prentice-Hall, Englewood Cliffs, New Jersey, see Chapter 4, "Structured Programming", p. 221-328.

Ledgard, H. F., L. J. Chmura (1978). FORTRAN with Style: Programming Proverbs, Rochelle Park, New Jersey, Hayden Book Co., 164 p.

McCracken, D. D. (1972a). A Guide to FORTRAN IV Programming, New York, John Wiley and Sons.

McCracken, D. D., G. M. Weinberg (1972b). How to Write a Readable FORTRAN Program, Datamation, October p. 73-76.

Roberts, R. V. (1969). The Publication of Scientific FORTRAN Programs, Computer Physics Communication, v. 1, p. 1-9.

Yourdon, E. (1975). Techniques of Program Structure and Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, See Chapter 4, "Structure Programming", p. 93-135.

Appendix A. Fortran Keywords

The following is a list of FORTRAN keywords and other character strings that should be avoided when naming quantities such as variables, programs, subroutines and arrays:

ACCESS	FALSE	PARAMETER
AND	FILE	PAUSE
ARRAYS	FMT	PDUMP
ASSIGN	FORM	PLOT
	FORMAT	PRECISION
BACKSPACE	FORMATTED	PRINT
BLANK	FUNCTION	PROGRAM
BLOCK		PUNCH
BUFFER	GE	PUT
	GET	
CALL	GOTO	READ
CHARACTER	GT	READMS
CLOSE		READEC
CLOSEM	IF	REAL
CLOSMS	IMPLICIT	REC
COMMON	INQUIRE	RECL
COMPLEX	INTEGER	RECOVR
CONTINUE	INTRINSICS	REMARK
		RETURN
DATA	LABEL	RETURNS
DATE	LOGICAL	REWIND
DECODE	LT	
DIMENSION		SAVE
DO		STATUS
DOUBLE	MAXREC	STOP
DUMP		SUBROUTINE
	NAME	SYMBOL
ELSE	NAMED	
ENCODE	NAMELIST	TAPE
END	NE	THEN
ENDFILE	NEXTREC	TIME
ENTRY	NOT	TRACE
EOF	NUMBER	TRUE
EQ		TYPE
EQUIVALENCE	OPEN	
ERR	OPENED	UNFORMATTED
ERRSET	OPENM	UNIT
EXIST	OPENMS	
EXIT	OR	WEOR
EXTERNAL	OVERLAY	WRITE
		WRITEC
		WRITMS

Appendix B. Basic External Functions

(From ANSI Standard X3.9-1966)

Basic External Function	Definition	Number of Arguments	Symbolic Name	Type of: Argument	Type of: Function
Exponential	e^a	1	EXP	Real	Real
		1	DEXP	Double	Double
		1	CEXP	Complex	Complex
Natural Logarithm	$\log_e(a)$	1	ALOG	Real	Real
		1	DLOG	Double	Double
		1	CLOG	Complex	Complex
Common Logarithm	$\log_{10}(a)$	1	ALOG10	Real	Real
			DLOG10	Double	Double
Trigonometric Sine	sin(a)	1	SIN	Real	Real
		1	DSIN	Double	Double
		1	CSIN	Complex	Complex
Trigonometric Cosine	cos(a)	1	COS	Real	Real
		1	DCOS	Double	Double
		1	CCOS	Complex	Complex
Hyperbolic Tangent	tanh(a)	1	TANH	Real	Real
Square Root	$(a)^{1/2}$	1	SQRT	Real	Real
		1	DSQRT	Double	Double
		1	CSQRT	Complex	Complex
Arctangent	arctan(a)	1	ATAN	Real	Real
		1	DATAN	Double	Double
	$\arctan(a_1/a_2)$	2	ATAN2	Real	Real
		2	DATAN2	Double	Double
Remaindering	$a_1 (\mathrm{mod}\ a_2)$	2	DMOD	Double	Double
Modulus		1	CABS	Complex	Real

*The function DMOD (a_1, a_2) is defined as $a_1 - [a_1/a_2]\, a_2$, where [x] is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as the sign of x.

Appendix C. Basic Intrinsic Functions

(From ANSI Standard X3.9-1966)

Intrinsic Function	Definition	Number of Arguments	Symbolic Name	Type of: Argument	Type of: Function
Absolute Value	$\lvert a \rvert$	1	ABS IABS DABS	Real Integer Double	Real Integer Double
Truncation	Sign of a times largest integer $< \lvert a \rvert$	1	AINT INT IDINT	Real Real Double	Real Integer Integer
Remaindering*	$a_1 (\mathrm{mod}\ a_2)$	2	AMOD MOD	Real Integer	Real Integer
Choosing Largest Value	Min $(a_1, a_2, \ldots)$	>2	AMAX0 AMAX1 MAX0 MAX1 DMAX1	Integer Real Integer Real Double	Real Real Integer Integer Double
Choosing Smallest Value	Min $(a_1, a_2, \ldots)$	>2	AMIN0 AMIN1 MIN0 MIN1	Integer Real Integer Double	Real Real Integer Double
Float	Conversion from integer to real	1	FLOAT	Integer	Real
Fix	Conversion from real to integer	1	IFIX	Real	Integer
Transfer of Sign	Sign of a_2 times $\lvert a_1 \rvert$	2	SIGN ISIGN DSIGN	Real Integer Double	Real Integer Double
Positive Difference	a_1 - Min (a_1, a_2)	2	DIM IDIM	Real Integer	Real Integer
Obtain Most Significant Part of Double Precision Argument		1	SNGL	Double	Real
Obtain Real Part of Complex Argument		1	REAL	Complex	Real
Obtain Imaginary Part of Complex Argument		1	AIMAG	Complex	Real
Express Single Precision Argument in Double Precision Form		1	DBLE	Real	Double
Express Two Real Arguments in Complex Form	$a_1 + a_2 \quad -1$	2	CMPLX	Real	Complex
Obtain Conjugate of a Complex Argument		1	CONJG	Complex	Complex

*The function MOD or AMOD(a_1, a_2) is defined as $a_1 - [a_1/a_2]\, a_2$ where $\lvert x \rvert$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as x.

Appendix D. Fortran Structures for Emulating Structured Programming Constructs

It is generally recognized that FORTRAN (X3.9-1966) is not a good language for structured programming (Yourdon, 1975; Jensen, 1979). Its major deficiencies are: (1) lack of block structures, as are available in languages such as Pascal and Algol; (2) lack of a nested IF-THEN-ELSE statement; and (3) lack of DO-WHILE and PERFORM-UNTIL statements.

Although the absence of these statements increases the difficulty of writing structured programs in FORTRAN, it should not be assumed that one cannot write structured programs in FORTRAN. One merely has to write control structures corresponding to those advocated by structured programming enthusiasts. Interestingly enough, however, to do so in FORTRAN requires the use of the GO TO statement, which at one time some structured programming enthusiasts were considering making illegal. Now it is generally agreed that the GO TO statement itself is not bad, but rather its uncontrolled use. One cannot be too upset with the original FORTRAN designers, because the language was around long before anyone even thought of structured programming. Some of the deficiencies of FORTRAN have been recognized and have been corrected in the more recent version of FORTRAN (X3.9-1978). For example, the revised language now includes the equivalent of an IF-THEN-ELSE statement. It still does not, however, include a DO-WHILE or PERFORM-UNTIL statement, nor does it include the block structure concepts of languages such as ALGOL and PASCAL. Inclusion of the latter would significantly change the whole design of FORTRAN and, in the author's opinion, block structure unlikely to be standar'ized in FORTRAN in the near future. This appendix describes the logical control structure diagrams, the equivalent structured programming pseudocode, and the FORTRAN code for implementing the following structured programming constructs: IF-THEN-ELSE, IF-ORIF-ELSE, CASE (two forms), POSIT (two forms), DO-WHILE, PERFORM-UNTIL, ESCAPE and CYCLE. The motivation for these constructs is described in considerable detail in Chapter 4 of Jensen and Tonies excellent book on software engineering (Jensen, 1979). It should be pointed out that some of the above forms are not necessarily advocated by all software engineers. In particular, the IF-ORIF-ELSE, POSIT, ESCAPE and CYCLE constructs were not described in Yourdon's (1975) book. Nevertheless, Jensen and Tonies present good arguments for their inclusion and thus they are mentioned here. It should also be pointed out that we have taken the liberty of adding another construct, the PERFORM-UNTIL (Form 2). This construct is equivalent to the traditional FORTRAN DO loop, which is so useful for indexing arrays and performing other counting operations. It was included here because the author felt it was unreasonable to request that programmers implement their looping operations with the FORTRAN equivalent code PERFORM-UNTIL (form 1), which uses the GO TO and IF statemens, when the standard FORTRAN DO loop could in many cases do the same task more concisely.

1. SEQUENCE

1.1 Logical Control Structure

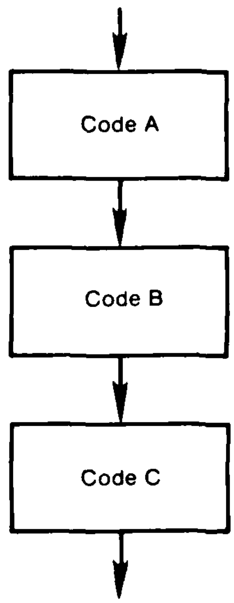


Figure D-1

Sequence Structure

1.2 Pseudocode

```
CODE A
CODE B
CODE C
```

1.3 FORTRAN Implementation

```
CODE A
CODE B
CODE C
```

2. IF-THEN-ELSE

2.1 Logical Control Structure

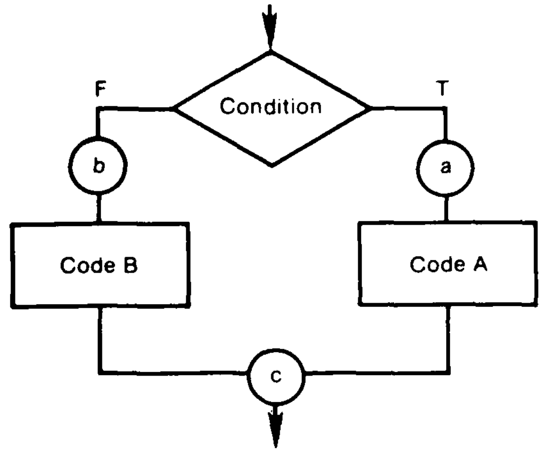


Figure D-2

IF-THEN-ELSE

Structure

2.2 Pseudocode

```
IF (condition) THEN
     CODE A
ELSE
     CODE B
ENDIF
```

2.3 FORTRAN Implementation

```
IF (condition) GO TO a
     CODE B
     GO TO c
a CONTINUE
     CODE A
c  CONTINUE
```

3.0 IF-or-IF-ELSE

3.1 Logical Control Structure

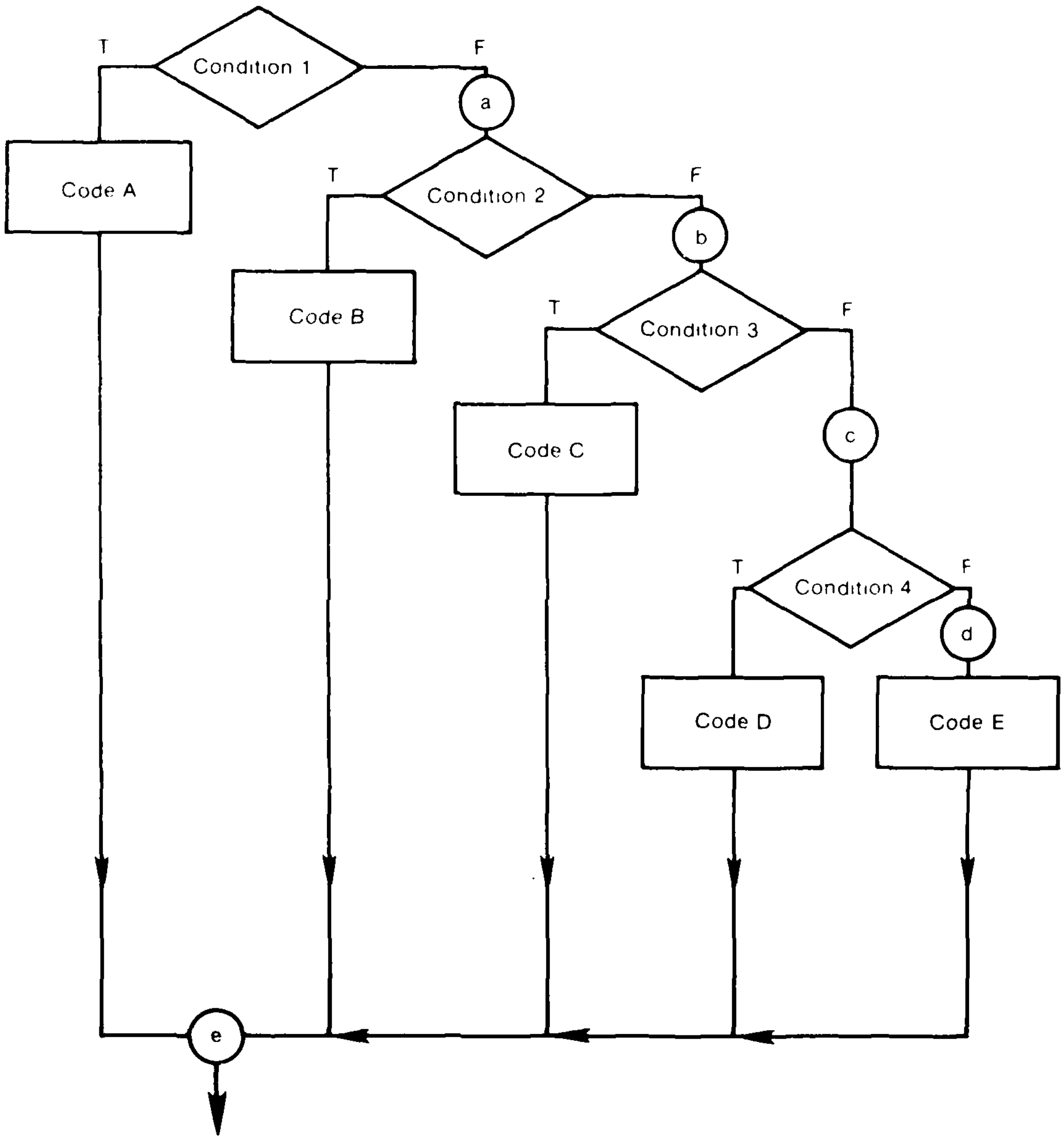


Figure D-3

IF-OR-IF-ELSE

Structure

3.2 Pseudocode

```
IF (condition 1)
     CODE A
ORIF (condition 2)
     CODE B
ORIF (condition 3)
     CODE C
ORIF (condition 4)
     CODE D
ELSE
     CODE E
ENDIF
```

3.3 FORTRAN Implementation

```
     IF (.NOT. Condition 1) GO TO a
     CODE A
     GO TO e
a CONTINUE
     IF (.NOT. Condition 2) GO TO b
     CODE B
     GO TO e
b CONTINUE
     IF (.NOT. Condition 3) GO TO c
     CODE C
     GO TO e
d CONTINUE
     IF (.NOT. Condition 4) GO TO d
     CODE D
     CODE E
e CONTINUE
```

4. CASE Statement - (Form 1)

4.1 Logical Control Structure

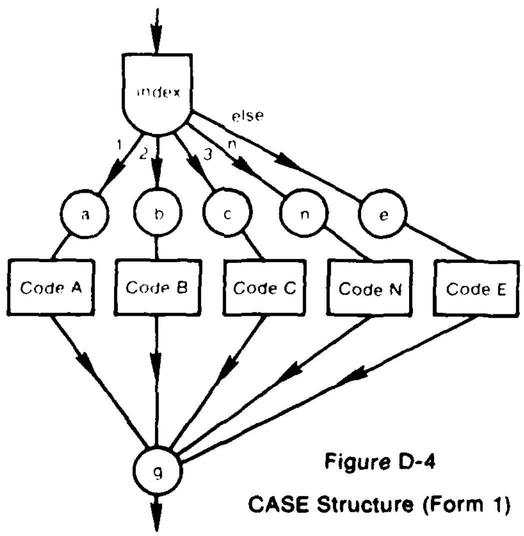


Figure D-4

CASE Structure (Form 1)

4.2 Pseudocode

```
CASE OF (index)
CASE (1)
     CODE A
CASE (2)
     CODE B
CASE (3)
     CODE C
CASE (N)
     CODE N
CASE ELSE
     CASE E
END CASE
```

4.3 FORTRAN Implementation

```
  IF (index .LT. 1 .OR. index
.GT. n) GO TO e
     GO TO (a, b, c, ...n), index
  a CONTINUE
     CODE A
     GO TO g
  b CONTINUE
     CODE B
     GO TO g
  c CONTINUE
     CODE C
     GO TO g
     . . .
     . . .
     . . .
  n CONTINUE
     CODE N
     GO TO g
  e CONTINUE
     CODE E
  g CONTINUE
```

5. CASE Statement (Form 2)

5.1 Logical Control Structure

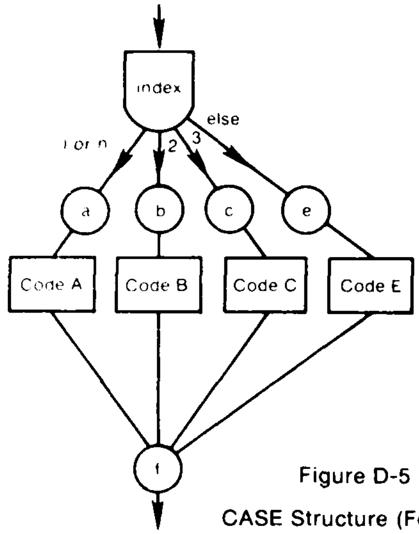


Figure D-5

CASE Structure (Form 2)

5.2 Pseudocode

```
CASE OF (index)
CASE (1,n)
    CODE A
CASE (2)
    CODE B
CASE (3)
    CODE C
CASE ELSE
    CODE E
END CASE
```

5.3 FORTRAN Implementation

```
IF (index .NE.1 .AND. index
.NE. n) GO TO b
        CODE A
      GO TO f
b  CONTINUE
      IF (index .NE. 2) GO TO c
        CODE B
      GO TO f
c CONTINUE
      IF (index .NE. 3) GO TO d
        CODE C
      GO TO f
d CONTINUE
        CODE E
f CONTINUE
```

6. POSIT - (Form 1)

6.1 Logical Control Structure

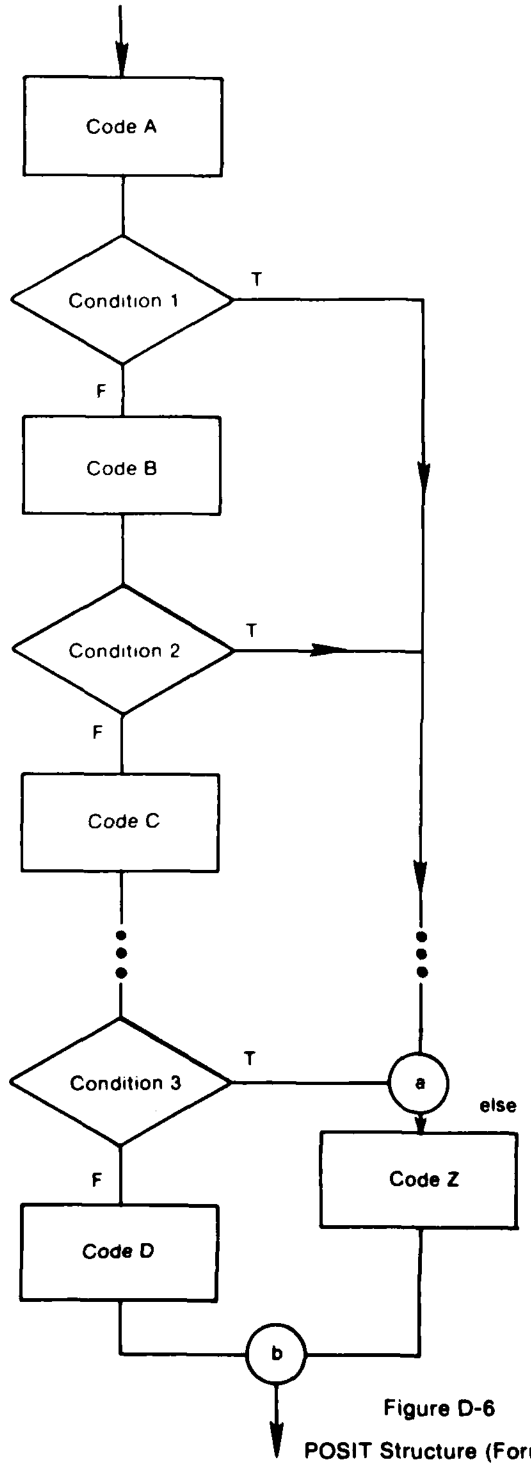


Figure D-6

POSIT Structure (Form 1)

6.2 Pseudocode

```
POSIT
     CODE A
QUIT POSIT IF (Condition 1)
     CODE B
QUIT POSIT IF (Condition 2)
     CODE C
       .
       .
       .
POSIT ELSE
     CODE Z
END POSIT
```

6.3 FORTRAN Implementation

```
      CODE A
      IF (Condition 1) GO TO a
        CODE B
      IF (Condition 2) GO TO a
        CODE C
      IF (Condition 3) GO TO a
        CODE D
      GO TO b
    a CONTINUE
         CODE Z
    b CONTINUE
```

7. POSIT - (Form 2)

7.1 Logical Control Structure

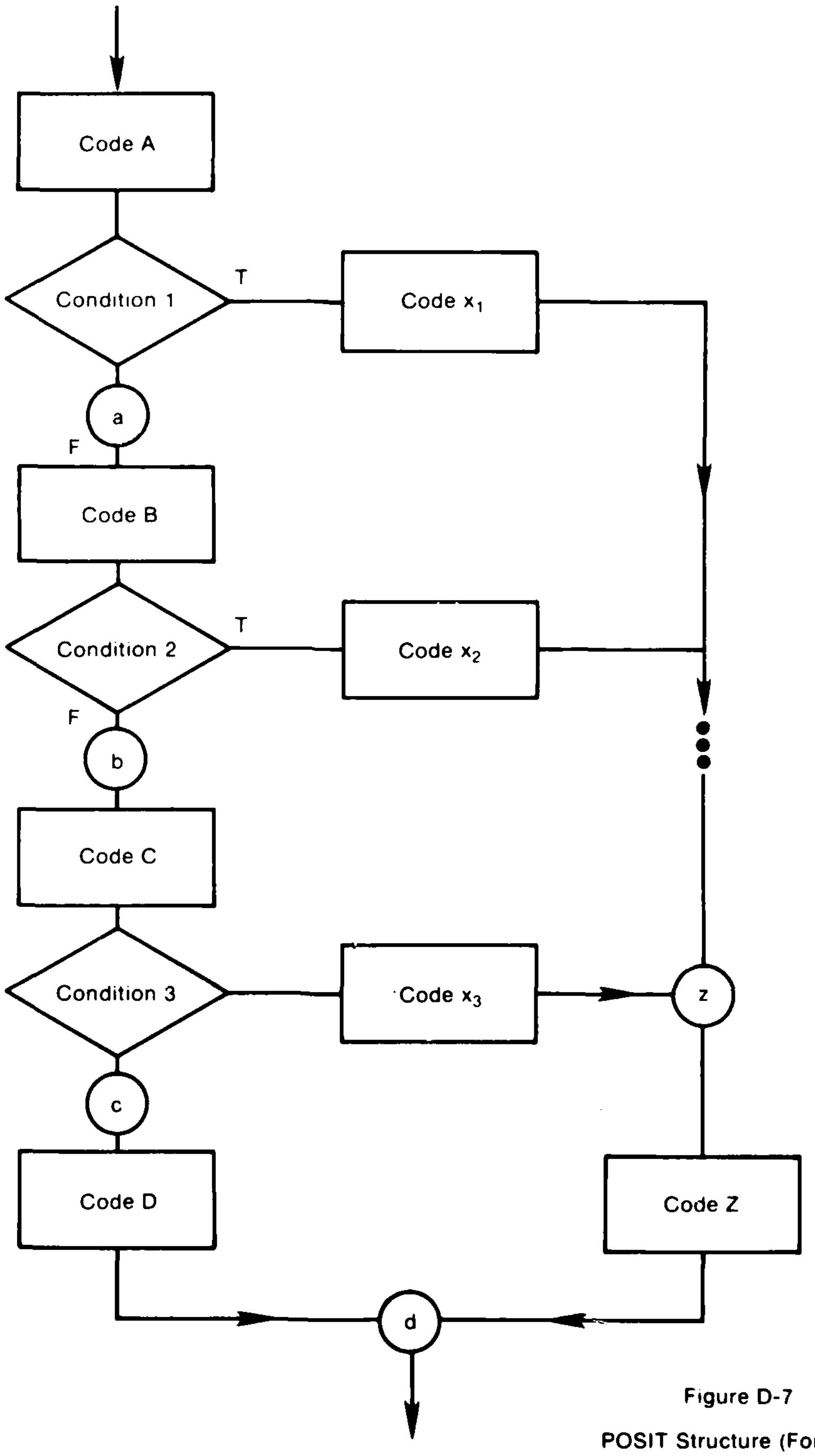


Figure D-7

POSIT Structure (Form 2)

7.2 Pseudocode

```
POSIT
     CODE A
IF (Condition 1)
     CODE X1
QUIT POSIT
ENDIF
     CODE  B
IF (Condition 2)
     CODE X2
QUIT POSIT
ENDIF
     CODE C
IF (Condition 3)
     CODE X3
QUIT POSIT
ENDIF
     CODE D
POSIT ELSE
     CODE Z
END POSIT
```

7.3 FORTRAN Implementation

```
     CODE A
  IF (.NOT. Condition 1) GO TO a
     CODE XI
     GO TO z
a CONTINUE
     CODE B
   IF (.NOT. Condition 2) GO TO b
     CODE X2
     GO TO z
b CONTINUE
     CODE C
  IF (.NOT. Condition 3)) GO TO c
     CODE X3
     GO TO z
c CONTINUE
     CODE D
     GO TO d
z CONTINUE
     Code Z
d CONTINUE
```

8. DO-WHILE

8.1 Logical Control Structure

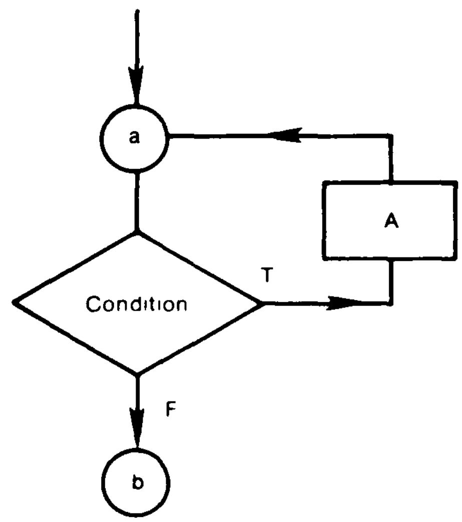


Figure D-8

DO-WHILE Structure

8.2 Pseudocode

```
WHILE (condition)
     CODE A
END WHILE
```

8.3 FORTRAN Implementation

```
a CONTINUE
     IF (.NOT. condition) GO TO b
       CODE A
     GO TO a
b CONTINUE
```

An alternative, but less popular, implementation which has the advantage of a positive test on the predicate is:

```
     GO TO a
c CONTINUE
       CODE A
a CONTINUE
     IF (condition) GO TO c
```

9. PERFORM-UNTIL (Form 1)

9.1 Logical Control Structure

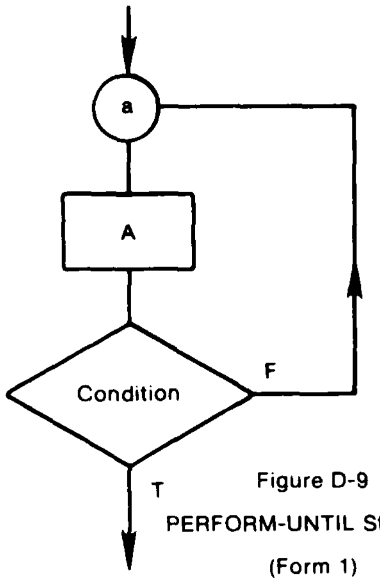


Figure D-9
PERFORM-UNTIL Structure
(Form 1)

9.2 Pseudocode

```
UNTIL (condition)
     CODE A
END UNTIL
```

9.3 FORTRAN Implementation

```
a CONTINUE
       CODE A
IF (.NOT. condition) GO TO a
```

An alternative implementation which has the advantage of a positive test on the predicate is:

```
     GO TO b
a CONTINUE
     IF (condition) GO TO d
b CONTINUE
       CODE A
     GO TO a
d CONTINUE
```

10. PERFORM-UNTIL (Form 2 - DO LOOP Equivalent)

10.1 Logical Control Structure

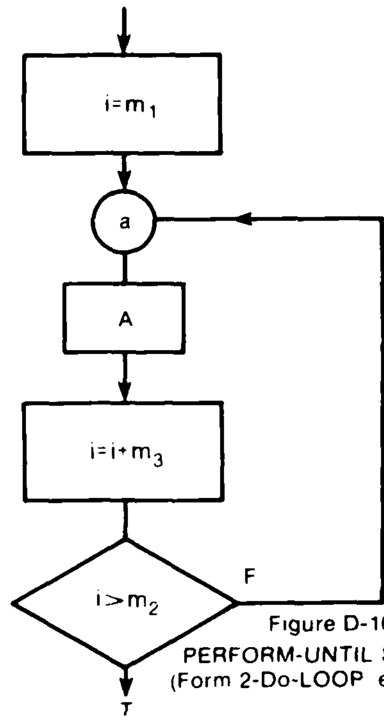


Figure D-10
PERFORM-UNTIL Structure
(Form 2-Do-LOOP equivalent)

10.2 Pseudocode

```
i=m1
UNTIL (i .GT. m2)
  CODE A
i=i+m3
END-UNTIL
```

10.3 FORTRAN Implementation

```
    DO a i=m1, m2, m3
          CODE A
  a CONTINUE
```

11. ESCAPE

The ESCAPE structure is an unconditional branch to the "outside" of its associated structure. If the exit is from an iterative loop, the branch would be to the outside of the loop.

This provides a mechanism for an easy exit from the interior of a set of nested iterative loops.

11.1 Logical Control Structure

Example of an escape in a DO-WHILE structure, The code B's Executed at the time of the escape

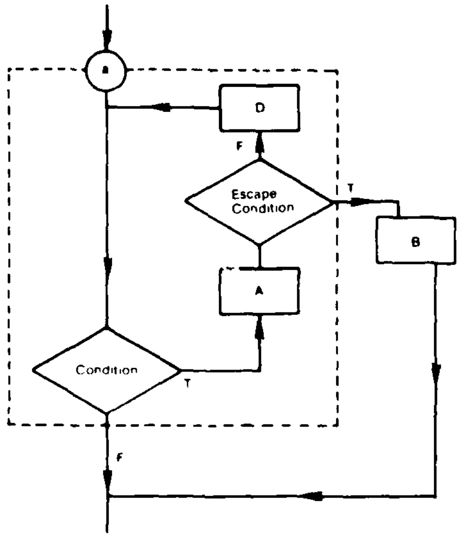


Figure D-11

ESCAPE-Structure

11.2 Pseudocode (Example)

```
S:  WHILE (condition)
      CODE A
    IF (escape-condition)
      CODE B
      ESCAPE WHILE S
    ENDIF
      CODE D
    END WHILE
```

11.3 FORTRAN Implementation

```
a CONTINUE
IF (.NOT. condition) GO TO b
  CODE A
IF (.NOT. escape-condition) GO TO d
  CODE B
  GO TO b
d CONTINUE
  CODE D
  GO TO a
b CONTINUE
```

12. CYCLE

The CYCLE structure is an unconditional branch to the condition controlling the next iteration, i.e. to the "inside" of the iteration loop. This provides a mechanism for easily by-passing code in the loop to advance to the next iteration.

12.1 Logical Control Structure

Example of a cycle in a DO-WHILE

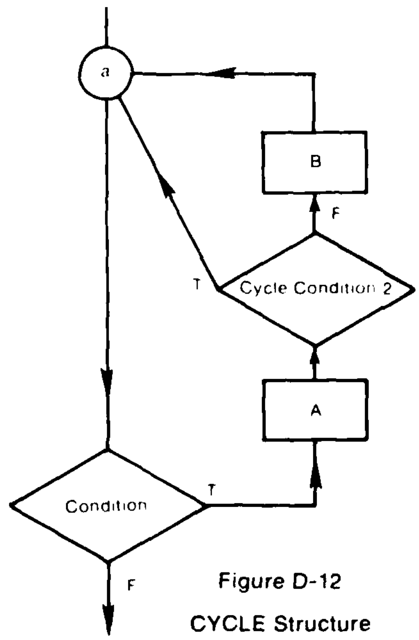


Figure D-12

CYCLE Structure

12.2 Pseudocode

```
C:  WHILE (condition)
      CODE A
    IF (cycle condition) CYCLE C
      CODE B
    END WHILE
```

12.3 FORTRAN Implementation

```
a CONTINUE
  IF (.NOT. condition) GO TO b
     CODE A
  IF (cycle-condition) GO TO a
     CODE B
   GO TO a
b CONTINUE
```

13. Example of a DO-WHILE Construct Incorporating ESCAPE and CYCLE

13.1 Logical Control Structure

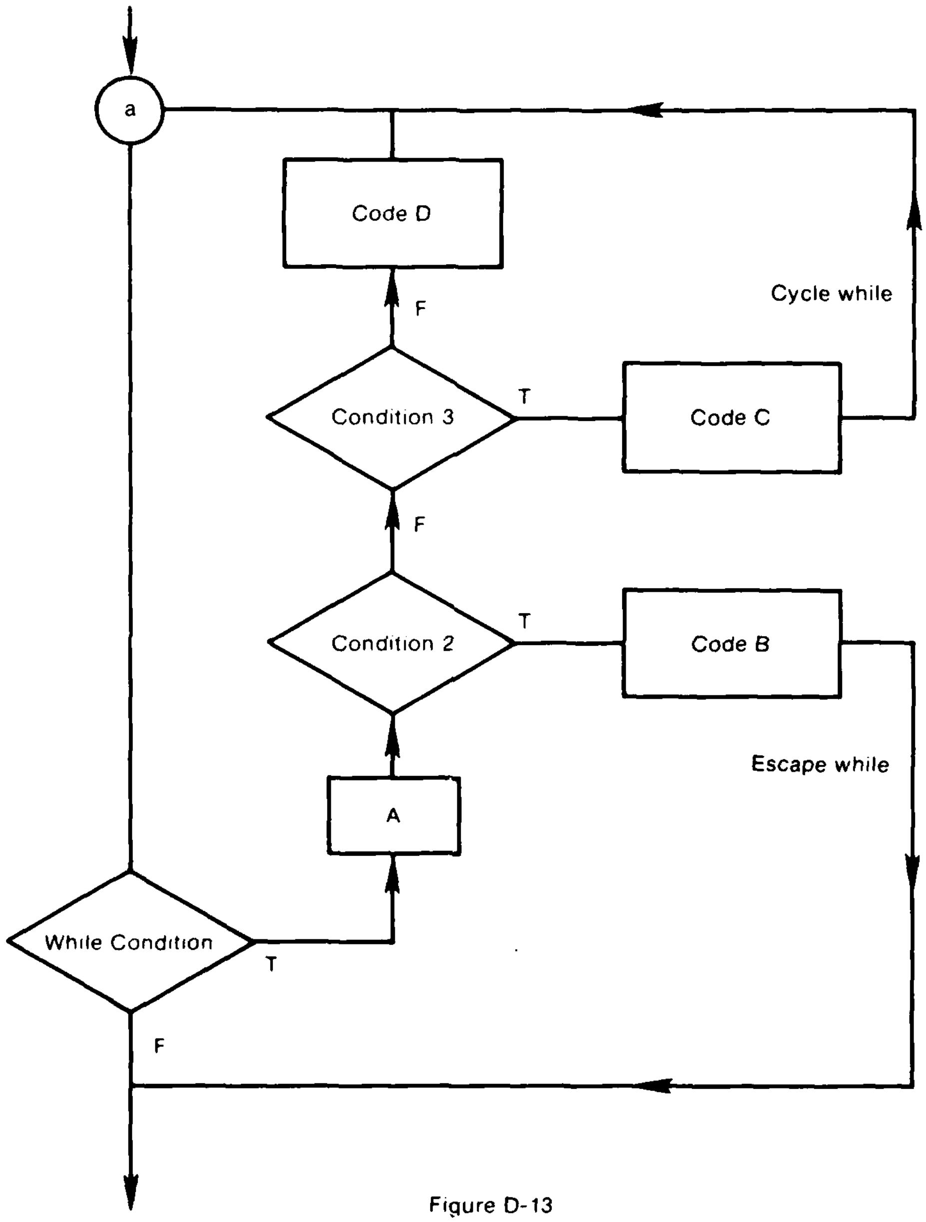


Figure D-13

DO-WHILE Construction

Incorporating ESCAPE and CYCLE

3.2 Pseudocode

```
S:  WHILE (while condition)
      CODE A
    IF (condition 2)
      CODE B
      ESCAPE WHILE S
    ENDIF
    IF (condition 3)
      CODE C
      CYCLE S
    ENDIF
      CODE D
    END WHILE S
```

3.3 FORTRAN Implementation

```
a CONTINUE
  IF (.NOT. while condition) GO TO d
      CODE A
  IF (.NOT. condition 2) GO TO e
      CODE B
    GO TO d
e CONTINUE
    IF (.NOT. condition 3) GO TO f
      CODE C
    GO TO a
f CONTINUE
      CODE D
    GO TO a
d CONTINUE
```

DISTRIBUTION LIST

COMMANDER 1
SECOND FLEET
FPO NEW YORK, NY 09501

COMMANDER 1
SIXTH FLEET
FPO NEW YORK, NY 09501

COMMANDER 1
ANTISUBMARINE WAR FORCE
U. S. SIXTH FLEET
FPO NEW YORK, NY 09501

COMMANDER 1
OCEANOGRAPHIC SYSTEM ATLANTIC
BOX 100
NORFOLK, VA 23511
ATTN: N3
CODE N34
CODE N36
CODE 012

COMMANDING OFFICER 1
NAVAL OCEAN RESEARCH & DEVELOPMENT
ACTIVITY
NSTL STATION, MS 39529

ATTN: CODE 110	1
CODE 125	1
CODE 115	1
CODE 300	1
CODE 320	1
CODE 321	1
CODE 322 (M. Clancy)	10
CODE 323	10
CODE 340	1
CODE 360	1
CODE 500	1
CODE 520 FILE	1

COMMANDER 1
NAVAL OCEANOGRAPHIC OFFICE
NSTL STATION, MS 39529

ATTN: CODE 7300	1
CODE 9000	1

ASSISTANT SECRETARY OF THE NAVY
(RESEARCH ENG. AND SYSTEM)
DEPARTMENT OF THE NAVY
WASHINGTON, DC 20350
ATTN: G. A. CANN 1

CHIEF OF NAVAL OPERATIONS
DEPARTMENT OF THE NAVY
WASHINGTON, DC 20350

ATTN: OP-02	1
OP-03	1
OP-05	1
OP-095	1
OP-096	1
OP-951	1
OP-952	1
OP-951F	1
OP-952D	1

HEADQUARTERS
NAVAL MATERIAL COMMAND
WASHINGTON, DC 20360
ATTN: MAT-0724 2

PROJECT MANAGER
ANTISUBMARINE WARFARE SYSTEM PROJ
DEPARTMENT OF THE NAVY
WASHINGTON, DC 20360
ATTN: PM-4 2

CHIEF OF NAVAL RESEARCH
800 NORTH QUINCY STREET
ARLINGTON, VA 22217

ATTN: CODE 100	1
CODE 102B	1
CODE 220	1
CODE 230	1
CODE 460	1
CODE 480	1

COMMANDER
NAVAL ELECTRONIC SYSTEMS COMMAND
NAVAL ELECTRONIC SYS COMMAND HDQRS
WASHINGTON, DC 20360

ATTN: PME-124	1
PME-124TA	1
PME-124/30	1
PME-124/40	1
PME-124/60	1
ELEX-320	1

COMMANDER
NAVAL SEA SYSTEMS COMMAND
NAVAL SEA SYS COMMAND HDQRS
WASHINGTON, DC 20362
ATTN: NSEA-06H1 1

DEPUTY UNDER SEC OF DEFENSE FOR 1
RESEARCH AND ENGINEERING
DEPARTMENT OF DEFENSE
WASHINGTON, DC 20361

COMMANDER 1
NAVAL OCEANOGRAPHY COMMAND
NSTL STATION, MS 39529

COMMANDER IN CHIEF
U.S. ATLANTIC FLEET
NORFOLK, VA 23511
ATTN: CODE 358 1

PLANNING SYSTEMS, INC.
7900 WESTPARK DRIVE
SUITE 600
MCLEAN, VA 22101
ATTN: R. KLINKNER 1
DR. R.S. CAVANAUGH 1
B.A. BRUNSON 1

SCIENCE APPLICATIONS, INC.
8400 WESTPARK DRIVE
MCLEAN, VA 22101
ATTN: DR. J.S. HANNA 1

TRACOR, INC.
1601 RESERCH BLVD.
ROCKVILLE, MD 20850
ATTN: J.T. GOTTWALD 1

TRW SYSTEMS GROUP
7600 COLSHIRE DRIVE
MCLEAN, VA 22101
ATTN: I.B. GEREBEN 1

UNIVERSITY OF TEXAS
APPLIED RESEARCH LABORATORIES
P.O. BOX 8029
AUSTIN, TX 78712
ATTN: G.E. ELLIS 1
L.D. HAMPTON 1
K.E. HAWKER 1
S.K. MITCHELL 1
S.G. PAYNE 1
J. SHOOTER 1

ANALYSIS AND TECHNOLOGY, INC.
TECHNOLOGY PARK P.O. BOX 220
NORTH STONINGTON, CT 06359
ATTN: S. ELAM 1

DAUBIN SYSTEMS CORP.
104 CRANDON BOULEVARD
SUITE 315
KEY BISCAYNE, FL 33149
ATTN: DR. S.C. DAUBIN 1

NAVAL OCEAN RESEARCH & DEVEL. ACT.
LIAISON OFFICE
800 NORTH QUINCY STREET
ARLINGTON, VA 22217
ATTN: CODE 130 1

COMMANDING OFFICER 1
NAVAL INTELLIGENCE SUPPORT CENTER
4301 SUITLAND ROAD
WASHINGTON, DC 20390

DEFENSE SYSTEMS, INC.
6110 EXECUTIVE BLVD. SUITE 320
ROCKVILLE, MD 20852
ATTN: G. JACOBS 5
J. LOCKLIN 1

THIRD WAVE SYSTEMS, INC.
P.O. BOX 7206
ST. PETERSBURG, FL 33734
ATTN: J.J. CORNYN 4

MR. BRUCE MENDENHALL 1
SAI
2999 MONTEREY-SALINAS HIGHWAY
MONTEREY, CA 93940

DR. GORDON WILLIAMS 1
SAI
1200 PROSPECT STREET
LAJOLLA, CA 92038

MR. GEORGE INNIS 1
SAI
1200 PROSPECT STREET
LAJOLLA, CA 92038

MR. KEN POLLAK 1
DATA INTEGRATION DEPT.
FNOC
NPGS ANNEX
MONTEREY, CA 93940

MS. BONNIE HUNTER DATA INTEGRATION DEPT. FNOC NPGS ANNEX MONTEREY, CA 93940	1
LCDR DUDLEY LEATH DATA INTEGRATION DEPT. FNOC NPGS ANNEX MONTEREY, CA 93940	1

DAT
ILM