AD-A031 458 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
STRUCTURED PROGRAMMING TRANSLATORS. VOLUME II. STRUCTRAN-1 USER--ETC(U)
AUG 76 D M ANDREWS, R A MELTON, R J URBAN F30602-75-C-0245
UNCLASSIFIED RADC-TR-76-253-VOL-2 NL

1 OF 1
AD A031458

END

DATE FILMED
12-76

RADC-TR-76-253, Vol II (of five)
Final Technical Report
August 1976

STRUCTURED PROGRAMMING TRANSLATORS
STRUCTRAN-1 User's Manual

General Research Corporation

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

This report consists of the following volumes:

I - Final Report
II - STRUCTRAN-1 User's Manual
III - STRUCTRAN-1 System Design and Implementation Manual
IV - STRUCTRAN-2 User's Manual
V - STRUCTRAN-2 System Design and Implementation Manual

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

This report has been reviewed and approved for publication.

APPROVED: Donald L. Mark

DONALD L. MARK
Project Engineer

APPROVED: Robert D. Krutz

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

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

MISSION
of
Rome Air Development Center

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-76-253, Vol II (of two)	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): STRUCTURED PROGRAMMING TRANSLATORS. Volume II. STRUCTRAN-1 User's Manual.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. May 75 - Jan 76
6. PERFORMING ORG. REPORT NUMBER: N/A	
7. AUTHOR(s): D. M. Andrews, R. A. Melton, R. J. Urban	8. CONTRACT OR GRANT NUMBER(s): F30602-75-C-0245
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Research Corporation, P. O. Box 3587, Santa Barbara CA 93105	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 32010314
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441	12. REPORT DATE: August 1976
	13. NUMBER OF PAGES: 21
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12) 26p.	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(16) AF-3201 (17) 320103

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

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Donald L. Mark (ISIS)

DMAAC Project Engineer: Ms. Opal Power

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

Structured Programming
Precompilers
Translators
Software Tools

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

The STRUCTRAN-1 User's Manual describes the various structured constructs and statement syntax of DMATRAN, a structured extension to FORTRAN. It also details the use of the STRUCTRAN-1 preprocessor, which translates DMATRAN into pure FORTRAN. Procedures for using either the machine independent or Univac 1108 adapted versions of STRUCTRAN-1 are included.

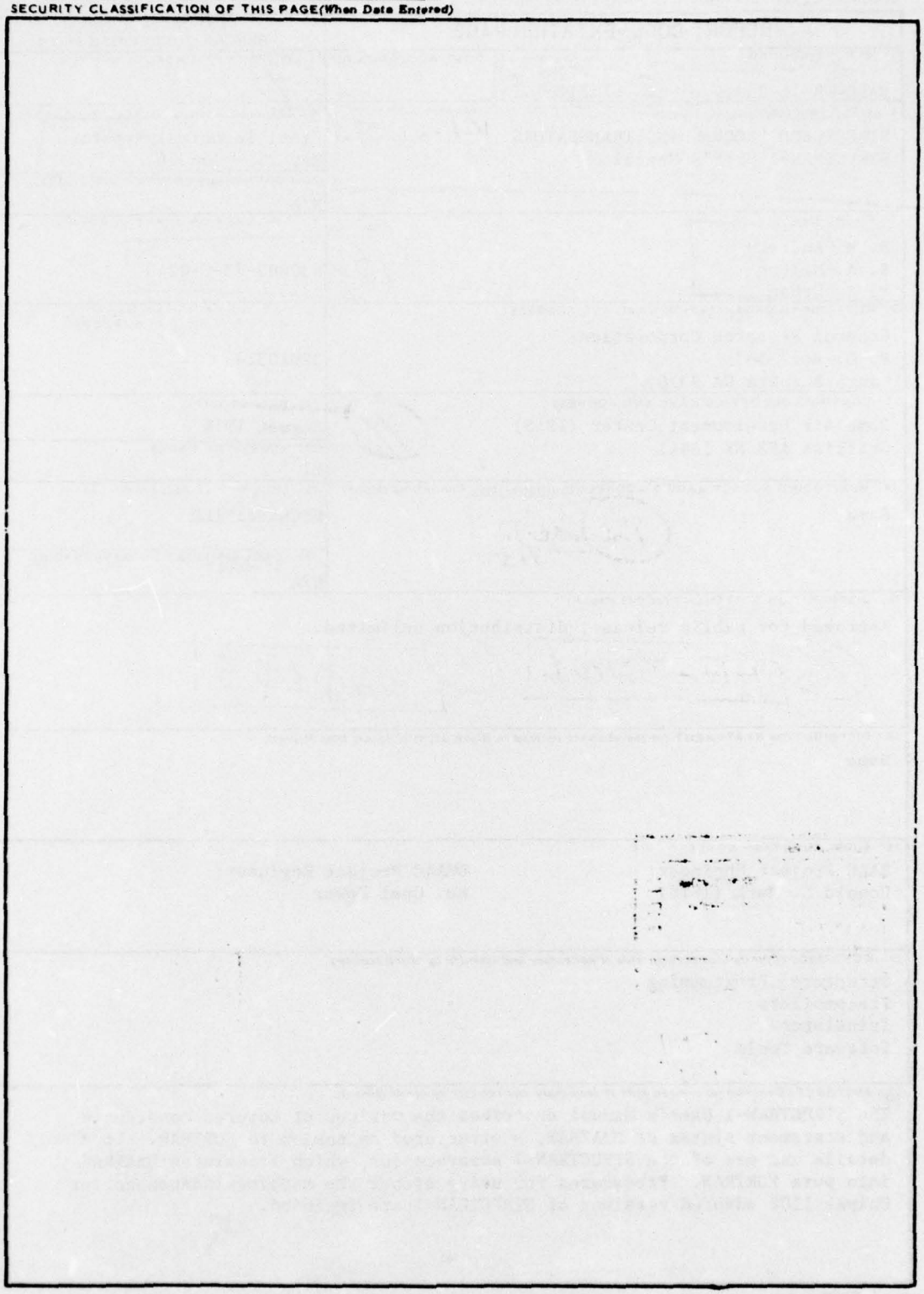
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

402 754

LB



CONTENTS

ILLUSTRATIONS

1 INTRODUCTION

The techniques of Structured Programming are finding increasing application in the computing community. Structured programs are, however, difficult to write in programming languages that do not have the statement types necessary to produce GOTO-free code. DMATRAN is a programming language that augments standard FORTRAN to permit its use as a basis for structured programming. Five structured programming statement forms are provided for use in FORTRAN-based software development. The STRUCTRAN-1 preprocessor translates DMATRAN into standard FORTRAN. It accepts as input modules containing both DMATRAN and FORTRAN statements, and produces as output indented pure-FORTRAN modules logically equivalent to the input modules. Features of STRUCTRAN-1 include automatic indentation on structured listings, comprehensive error processing to warn of improper statement forms, and warnings to indicate the presence of unstructured FORTRAN control statements. The STRUCTRAN-1 preprocessor is also compatible with the output of STRUCTRAN-2, a system which converts unstructured FORTRAN programs into structured form in DMATRAN.

The structured-programming statement forms implemented in STRUCTRAN-1 are described in Sec. 2, with typical examples of FORTRAN modules and corresponding DMATRAN modules. The use of the computer-independent version of STRUCTRAN-1 is discussed in Sec. 3; Sec. 3.5 elaborates on the STRUCTRAN-1 error diagnostics. Appendix A describes the use of STRUCTRAN-1 on the Univac 1108. STRUCTRAN-1 guidelines are summarized in Sec. 4.

2 DMATRAN CONSTRUCTS

DMATRAN replaces FORTRAN control statements with the following control statement constructs:

- IF...THEN...ELSE...END IF--provides block structuring of conditionally executable sequences of statments
- DO WHILE...END WHILE--permits iteration of a code segment while a specified condition remains true
- CASE OF...CASE...CASE ELSE...END CASE--allows multiple choices for selecting program action
- DO UNTIL...END UNTIL--permits iteration until a specified condition becomes true
- BLOCK<name>...END BLOCK (and corresponding INVOKE<NAME> statement)--provide for top-down programming and internal subroutines

These statement forms can be intermixed with standard FORTRAN non-control statements in the text stream which is processed by the STRUCTRAN-1 preprocessor. DMATRAN statements are converted by the preprocessor to their FORTRAN equivalents, and the resulting file can be compiled by the FORTRAN compiler in the normal manner.

A structured GOTO-free program has a highly visible form which reveals its intended function more readily than a FORTRAN program containing GOTO statements, which has no apparent form. Well-defined blocks of code are executed in a sequential, top-down manner. The possible sequences of blocks which can be executed are well-defined. The following simple examples illustrate these concepts.

2.1 IF...THEN...ELSE...END IF

The general form of the IF construct consists of three DMATRAN statements:

```
IF(<logical-expression>) THEN
      <block of statements>
ELSE
      <block of statements>
END IF
```

where <logical-expression> is any legal FORTRAN logical expression. The <logical-expression> is evaluated, and if it is true the statements following the IF are executed until the ELSE is reached, where control passes to the first statement after the END IF. The ELSE is optional. If the ELSE is absent and the <logical-expression> is false, control passes to the first statement after the END IF. If the ELSE is present and the <logical-expression> is false, control passes to the statement following the ELSE so that the

statements after the ELSE are executed. The unindented source input for an IF-construct is shown in Fig. 2.1a. The indentation performed by STRUCTRAN-1 is indicated in Fig. 2.1b.

```
IF (IN.GE.10) THEN
OUT=0
ELSE
OUT=OUT+1
END IF
```

(a)

DMATRAN Source Input

```
IF (IN.GE.10) THEN
.  OUT=0
ELSE
.  OUT=OUT+1
END IF
```

(b)

STRUCTRAN-1 Indented Listing

Figure 2.1. DMATRAN IF...THEN...ELSE...END IF Construct

2.2 DO WHILE...END WHILE

The general form of the DO WHILE construct consists of two DMATRAN statements:

```
DO WHILE(<logical-expression>)
      <block of statements>
END WHILE
```

where <logical-expression> is any legal FORTRAN logical expression. The DO WHILE construct represents an iteration in which execution occurs in the following manner:

(1) The value of <logical-expression> is found: if true, the statements contained within the DO WHILE block are executed; if false, control passes to the statement immediately following the END WHILE.

(2) If the statements within the DO WHILE block have been executed, the value of <logical-expression> is checked again, with the same consequences as in (1).

The iterative block in the DO WHILE...END WHILE may be executed zero or more times. In general it is necessary to initialize the loop control variable

before entering the WHILE construct, and also necessary to modify it within the WHILE construct. Figure 2.2a demonstrates the unindented source input for a WHILE construct; the corresponding indented listing produced by STRUCTRAN-1 is shown in Fig. 2.2b.

```
I=1
DO WHILE (IN(I).NE.OUT .AND. I.LE.50)
I=I+1
END WHILE
RETURN
END
```

(a) DMATRAN Source Input

```
I=1
DO WHILE (IN(I).NE.OUT .AND. I.LE.50)
.  I=I+1
END WHILE
```

(b) STRUCTRAN-1 Indented Listing

Figure 2.2. DMATRAN DO WHILE...END WHILE Construct

The IF construct and the DO WHILE construct are sufficient to express the control portion of any algorithm which can be implemented in FORTRAN. However, for greatest convenience in implementation of software systems with structured programming techniques, some additional statement forms are highly desirable. The CASE construct is a selection structure and the DO UNTIL is an iteration structure, while the BLOCK construct enhances modularity.

2.3 CASE OF...CASE...CASE ELSE...END CASE

The CASE statement provides a way to select which group of statements will be executed. The general form of the CASE construct consists of the following DMATRAN statements:

```
CASE OF(<integer-expression>)
CASE(<i>)
      <block of statements>
CASE(<j>)
      <block of statements>
CASE ELSE
      <block of statements>
END CASE
```

<i> and <j> represent integers of positive value. They may be in any order, and there is no limit to how many integers may be listed.

The <integer-expression> is computed, and if any of the specified integers in the CASE list are equal to the value of the expression, then the transfer of control is to the statements which follow that particular CASE. If there is no such CASE, and the CASE ELSE statement is present, then the block of statements following the CASE ELSE is executed; otherwise, no block is executed. If there are two CASE statements with the same CASE index, the first occurring one is executed (if the CASE expression has that value). After the block of statements selected has been executed, control transfers to the statement after the END CASE.

Figure 2.3a is the unindented source input for a CASE construct. The indented listing produced by STRUCTRAN-1 is shown in Fig. 2.3b.

```
CASE OF (IN)
CASE(10)
OUT=IN
CASE(15)
OUT=IN-3
CASE ELSE
OUT=IN*10
END CASE
RETURN
END
```

(a) DMATRAN Source Input

```
CASE OF (IN)
CASE(10)
.  OUT=IN
CASE(15)
.  OUT=IN-3
CASE ELSE
.  OUT=IN*10
END CASE
RETURN
END
```

(b) STRUCTRAN-1 Indented Listing

Figure 2.3. DMATRAN CASE OF...CASE...CASE ELSE...END CASE Construct

2.4 DO UNTIL...END UNTIL

The general form of the DO UNTIL construct consists of two DMATRAN statements:

```
DO UNTIL(<logical-expression>)
      <block of statements>
END UNTIL
```

The statements enclosed within the DO UNTIL and the END UNTIL are always executed once. Then the <logical-expression> is evaluated and, if false, iteration and evaluation of the expression continue until it is true. At that time execution of the statements following the END UNTIL begins.

Figures 2.4a and 2.4b indicate the DMATRAN source input and corresponding STRUCTRAN-1 indented listing for a DO UNTIL construct. After completion of the DO UNTIL iteration, J will have the value 16 and I the value 11. It is important to note that when using DO WHILE or DO UNTIL constructs, the iteration variable must be initialized before entering the iteration and modified within the iteration.

```
I=1
J=6
DO UNTIL (I.GT.10)
OUT(J)=IN(I)
I=I+1
J=J+1
END UNTIL
RETURN
END
```

(a) DMATRAN Source Input

```
I=1
J=6
DO UNTIL (I.GT.10)
.  OUT(J)=IN(I)
.  I=I+1
.  J=J+1
END UNTIL
```

(b) STRUCTRAN-1 Indented Listing

Figure 2.4. DMATRAN DO UNTIL...END UNTIL Construct

2.5 BLOCK...END BLOCK AND INVOKE

The constructs described in the preceding paragraphs allow most programming tasks to be done in a well-structured manner. However, they do not always permit top-down programming. To implement this method, one must be able to refer to an action (such as "compute array element") before the code for it is actually available.

The usual method for doing this is calling subroutines. However, subroutines have certain disadvantages. The overhead involved in calling them is often high. Additionally, those variables used in both a calling routine and a called subroutine must either be placed in COMMON or passed as parameters. those variables used in both a calling routine and a called subroutine must either be placed in COMMON or passed as parameters.

In many cases, a subroutine uses only variables which are already in the routine which calls it. Use of a subroutine internal to the calling routine eliminates the need for any mechanism (such as parameters or COMMON blocks) for referring to the variables required.

A facility for creating and using this type of subroutine has been added to DMATRAN. This construct is called a BLOCK which may be defined as an internal, parameterless procedure with all variables global. A BLOCK can be called only from the individual routine (main program, subroutine, or function) in which it is compiled; it cannot be called from an external routine, nor can it be passed as a parameter to another routine. A BLOCK is simply a segment of the code of the routine which contains it. The BLOCK is exercised only if it is invoked.

The general form of a BLOCK construct consists of two DMATRAN statements:

```
BLOCK(<block-name>)
      <statements>
END BLOCK
```

where <block-name> is any string of characters (e.g., COMPUTE.INDEX or PRINT-CURRENT-STATUS). The name of a BLOCK may be arbitrarily long, so that the name can have mnemonic significance. However, the first six characters must be unique.

A BLOCK is called by an INVOKE statement, whose format is:

```
INVOKE(<block-name>)
```

When an INVOKE statement is executed, control is transferred to the first statement in the BLOCK; when the END BLOCK is reached, control goes to the statement following the INVOKE of the BLOCK. Though BLOCKs can be nested (one BLOCK completely inside of another), no recursion is allowed in the calling of BLOCKs (i.e., a BLOCK cannot invoke itself). Also, the name of a BLOCK is known throughout the entire routine in which it is contained.

The following are examples of the two major uses of the BLOCK construct:

Example 1: Top-Down Programming

```
I=1
DO UNTIL (I .GT. N)
      J=1
      DO UNTIL (J .GT. N)
            INVOKE(COMPUTE.ARRAY.ELEMENT)
            J=J+1
      END UNTIL
      I=I+1
END UNTIL
```

and, at some place later in the same routine:

```
BLOCK(COMPUTE.ARRAY.ELEMENT)
      code to compute A(I,J)
      A(I,J) = value computed
      J=J+1
END BLOCK
```

The use of a BLOCK construct enhances readability and understandability of the program.

Example 2: Internal Subroutine

S_1 and S_2 in the following code represent two sets of statements. The use of a BLOCK in Method 2 below eliminates the need for duplicating code.

Method 1:

```
IF(A) THEN
      IF(C) THEN
            S1
      ELSE
            S2
      END IF
ELSE
      IF(D) THEN
            S2
      ELSE
            S1
      END IF
END IF
```

Method 2:

```
IF(A) THEN
      IF(C) THEN
            INVOKE(BLOCK-A)
      ELSE
            INVOKE(BLOCK-B)
      END IF
ELSE
      IF(D) THEN
            INVOKE(BLOCK-B)
      ELSE
            INVOKE(BLOCK-A)
      END IF
END IF
```

where the BLOCKs are defined as:

```
BLOCK(BLOCK-A)
      S1
END BLOCK
```

and

```
BLOCK(BLOCK-B)
      S2
END BLOCK
```

There is a maximum of 20 BLOCKs per module with a limit of 15 INVOKEs for any one BLOCK. The overhead in time and space involved in using BLOCKs is less than that of using subroutine calls.

Figure 2.5a indicates the DMATRAN source input for BLOCK constructs and related INVOKE statements. The indented listing produced by STRUCTRAN-1 is shown in Fig. 2.5b.

```
      INVOKE (COMPUTE AREA)
      INVOKE (PRINT AREA)
      BLOCK (COMPUTE AREA)
      AREA=LENGTH*WIDTH
      END BLOCK
      BLOCK (PRINT AREA)
      WRITE (6,1)AREA
    1 FORMAT (10X,120)
      END BLOCK
```

(a) DMATRAN Source Input

```
      INVOKE (COMPUTE AREA)
      INVOKE (PRINT AREA)
      BLOCK (COMPUTE AREA)
      .  AREA=LENGTH*WIDTH
      END BLOCK
      BLOCK (PRINT AREA)
      .  WRITE (6,1)AREA
    1 .  FORMAT (10X,120)
      END BLOCK
```

(b) STRUCTRAN-1 Indented Listing

Figure 2.5. DMATRAN BLOCK...END BLOCK and INVOKE Constructs

3 USING STRUCTRAN-1

3.1 STRUCTRAN-1 INPUT

Figure 3.1 illustrates a DMATRAN source program ready for input to the STRUCTRAN-1 preprocessor. The DMATRAN source code has not been manually indented in order to avoid the problem of updating indentation levels as the program is modified. The first card of the DMATRAN text sets optional parameters (reference page 13). In this case a blank card indicates that default parameter settings will be used. The last card of the STRUCTRAN-1 input must contain "END" in columns 1 through 3. More than one module may be processed in each STRUCTRAN-1 run. Site dependent modifications may alter the input format indicated in Fig. 3.1. The Univac 1108 requirements are described in Appendix A.

```
      <BLANK CARD>
      SUBROUTINE EXAMPL (INFO,LENGTH)                                  EXAMPL1
C                                                                      EXAMPL2
C     ILLUSTRATION OF DMATRAN SYNTAX                                   EXAMPL3
C                                                                      EXAMPL4
      IF (INFO.LE.10 .AND. LENGTH.GT.0)THEN                            EXAMPL5
      INFO=INFO+10                                                     EXAMPL6
      ELSE                                                             EXAMPL7
      LENGTH=50                                                        EXAMPL8
      END IF                                                           EXAMPL9
      CASE OF (INFO+6)                                                 EXAMPL10
      CASE (14)                                                        EXAMPL11
      LENGTH=LENGTH-INFO                                               EXAMPL12
      CASE (17)                                                        EXAMPL13
      DO WHILE (INFO.LT.20)                                            EXAMPL14
      DO UNTIL (LENGTH.LE.INFO)                                        EXAMPL15
      INVOKE (COMPUTE LENGTH)                                          EXAMPL16
      IF (LENGTH.GE.30) THEN                                           EXAMPL17
      INVOKE (PRINT-RESULTS)                                           EXAMPL18
      END IF                                                           EXAMPL19
      END UNTIL                                                        EXAMPL20
      INFO=INFO+1                                                      EXAMPL21
      END WHILE                                                        EXAMPL22
      CASE ELSE                                                        EXAMPL23
      DO WHILE (LENGTH.GT.0)                                           EXAMPL24
      INVOKE (COMPUTE LENGTH)                                          EXAMPL25
      END WHILE                                                        EXAMPL26
      END CASE                                                         EXAMPL27
      BLOCK (PRINT-RESULTS)                                            EXAMPL28
      WRITE (6,1)INFO,LENGTH                                           EXAMPL29
    1 FORMAT (10X,I5,20X,I5)                                           EXAMPL30
      END BLOCK                                                        EXAMPL31
      BLOCK (COMPUTE LENGTH)                                           EXAMPL32
      LENGTH = LENGTH -10                                              EXAMPL33
      END BLOCK                                                        EXAMPL34
      RETURN                                                           EXAMPL35
      END                                                              EXAMPL36
END
```

Figure 3.1. STRUCTRAN-1 Input

3.2 STRUCTRAN-1 INDENTED LISTING

Figure 3.2 illustrates the automatically indented DMATRAN listing which resulted from processing the input shown in Fig. 3.1. The heading contains information from the first card of the module being processed, as well as the page number. The leftmost column of numbers refers to the successive cards of the DMATRAN source deck. The nesting depth of each statement is indicated in parentheses. Structural visibility is enhanced by connecting related DMATRAN statements with vertical dots. The dots make it easier to trace paths through the program, help identify the card number for a given line, and aid in debugging improperly formed DMATRAN control structures. Structural errors are indicated by error diagnostics (see page 15) in the DMATRAN listing. Sequence information on the original DMATRAN source cards is included in the STRUCTRAN-1 listing.

```
NO. (..)=NESTING DEPTH          SUBROUTINE EXAMPL (INFO,LENGTH)  PAGE     1

  1             SUBROUTINE EXAMPL (INFO,LENGTH)                    EXAMPL1
  2       C                                                        EXAMPL2
  3       C          ILLUSTRATION OF DMATRAN SYNTAX
  4       C                                                        EXAMPL4
  5             IF (INFO.LE.10 .AND. LENGTH.GT.0)THEN              EXAMPL5
  6 ( 1)        . INFO=INFO+10                                     EXAMPL6
  7             ELSE                                               EXAMPL7
  8 ( 1)        . LENGTH=50                                        EXAMPL8
  9             END IF                                             EXAMPL9
 10             CASE OF (INFO+6)                                   EXAMPL10
 11             CASE (14)                                          EXAMPL11
 12 ( 1)        . LENGTH=LENGTH-INFO                               EXAMPL12
 13             CASE (17)                                          EXAMPL13
 14 ( 1)        . DO WHILE (INFO.LT.20)                            EXAMPL14
 15 ( 2)        . . DO UNTIL (LENGTH.LE.INFO)                      EXAMPL15
 16 ( 3)        . . . INVOKE (COMPUTE LENGTH)                      EXAMPL16
 17 ( 3)        . . . IF (LENGTH.GE.30) THEN                       EXAMPL17
 18 ( 4)        . . . . INVOKE (PRINT-RESULTS)                     EXAMPL18
 19 ( 3)        . . . END IF                                       EXAMPL19
 20 ( 2)        . . END UNTIL                                      EXAMPL20
 21 ( 2)        . . INFO=INFO+1                                    EXAMPL21
 22 ( 1)        . END WHILE                                        EXAMPL22
 23             CASE ELSE                                          EXAMPL23
 24 ( 1)        . DO WHILE (LENGTH.GT.0)                           EXAMPL24
 25 ( 2)        . . INVOKE (COMPUTE LENGTH)                        EXAMPL25
 26 ( 1)        . END WHILE                                        EXAMPL26
 27             END CASE                                           EXAMPL27
 28             BLOCK (PRINT-RESULTS)                              EXAMPL28
 29 ( 1)        . WRITE (6,1)INFO,LENGTH                           EXAMPL29
 30 ( 1)      1 . FORMAT (10X,I5,20X,I5)                           EXAMPL30
 31             END BLOCK                                          EXAMPL31
 32             BLOCK (COMPUTE LENGTH)                             EXAMPL32
 33 ( 1)        . LENGTH = LENGTH -10                              EXAMPL33
 34             END BLOCK                                          EXAMPL34
 35             RETURN                                             EXAMPL35
 36             END                                                EXAMPL36
```

Figure 3.2. STRUCTRAN-1 Input Indented Listing

3.3 FORTRAN OUTPUT

Figure 3.3 illustrates the translated FORTRAN version of Fig. 3.1 produced by the STRUCTRAN-1 preprocessor. This may be compiled and executed in the normal manner. The translated FORTRAN is indented to reflect the structure of the original DMATRAN program. The sequence information in columns 73 through 80 refers back to the card number of the original DMATRAN statement. This aids in relating FORTRAN error diagnostics to the DMATRAN source code to be modified. The DMATRAN source code (not the resultant FORTRAN) is the code which should be modified.

3.4 DEFAULT PARAMETER MODIFICATION

The first card input to STRUCTRAN-1 can be used to modify any of the default parameter settings which are not appropriate. A blank field on this card causes the default for the corresponding parameter to be used. This card is always read from FORTRAN unit 5, even though the first parameter may specify that the DMATRAN source input unit is other than 5.

COLUMN (Right Justified Within Column)	PARAMETER	DEFAULT
1-5	STRUCTRAN-1 input unit (DMATRAN source)	5
6-10	STRUCTRAN-1 listing unit (indented listing)	6
11-15	Translated FORTRAN unit (to be compiled)	2
16-20	STRUCTRAN-1 error diagnostics unit	6
21-25	Whether to perform automatic indentation for DMATRAN indented listing (1 = YES; 0 = NO)	1
26-30	Characters per indent level	3
31-35	Whether to indent comments (1 = YES; 0 = NO)	0
36-40	Whether to left-adjust all statements (1 = YES; 0 = NO)	1
41-45	Lines per page	57
46-50	Initial generated label	99998
51-55	Characters per line on indented listing	132
56-60	Whether to include comments in FORTRAN file (1 = YES; 0 = NO)	0

For example, a 4 in column 5 of the first input card will cause the DMATRAN source to be read from unit 4, while all other parameters retain their default values.

```
      SUBROUTINE EXAMPL (INFO,LENGTH)                                    1*
      IF (INFO.LE.10 .AND. LENGTH.GT.0) GO TO 99998                      5*
      GO TO 99997                                                        5*
99998 CONTINUE                                                           5*
        INFO=INFO+10                                                     6*
      GO TO 99996                                                        7*
99997 CONTINUE                                                           7*
        LENGTH=50                                                        8*
99996 CONTINUE                                                           9*
      I99995=INFO+6                                                     10*
      IF (I99995.NE.(14)) GO TO 99993                                   11*
        LENGTH=LENGTH-INFO                                              12*
      GO TO 99994                                                       13*
99993 CONTINUE                                                          13*
      IF (I99995.NE.(17)) GO TO 99992                                   13*
99991   IF (INFO.LT.20) GO TO 99990                                     14*
        GO TO 99989                                                     14*
99990   CONTINUE                                                        14*
          GO TO 99988                                                   15*
99987     IF (LENGTH.LE.INFO) GO TO 99986                               15*
99988     CONTINUE                                                      15*
            ASSIGN 99983 TO I99984                                      16*
            GO TO 99984                                                 16*
99983       CONTINUE                                                    16*
            IF (LENGTH.GE.30) GO TO 99982                               17*
            GO TO 99981                                                 17*
99982       CONTINUE                                                    17*
              ASSIGN 99979 TO I99980                                    18*
              GO TO 99980                                               18*
99979         CONTINUE                                                  18*
99981       CONTINUE                                                    19*
          GO TO 99987                                                   20*
99986     CONTINUE                                                      20*
99985     CONTINUE                                                      20*
          INFO=INFO+1                                                   21*
        GO TO 99991                                                     22*
99989   CONTINUE                                                        22*
      GO TO 99994                                                       23*
99992 CONTINUE                                                          23*
99978   IF (LENGTH.GT.0) GO TO 99977                                    24*
        GO TO 99976                                                     24*
99977   CONTINUE                                                        24*
          ASSIGN 99975 TO I99984                                        25*
          GO TO 99984                                                   25*
99975     CONTINUE                                                      25*
        GO TO 99978                                                     26*
99976   CONTINUE                                                        26*
99994 CONTINUE                                                          27*
      GO TO 99974                                                       28*
99980 CONTINUE                                                          28*
        WRITE (6,1)INFO,LENGTH                                          29*
    1   FORMAT (10X,I5,20X,I5)                                          30*
      GO TO I99980,(99979)                                              31*
99974 CONTINUE                                                          31*
      GO TO 99973                                                       32*
99984 CONTINUE                                                          32*
        LENGTH = LENGTH -10                                             33*
      GO TO I99984,(99983,99975)                                        34*
99973 CONTINUE                                                          34*
      RETURN                                                            35*
      END                                                               36*
```

Figure 3.3. STRUCTRAN-1 Translated FORTRAN

3.5 ERROR MESSAGES

Error messages generated by STRUCTRAN-1 are identified by number only and are output to the same unit as the structured listing (default option is 6). For example, the following error message might be printed: "ERROR 321." An explanation of the meaning of STRUCTRAN-1 error messages is presented in Table 3.1.

TABLE 3.1

STRUCTRAN-1 ERROR MESSAGE DEFINITIONS

Error Number	Message Interpretation
1	Non-comment with punch in Col. 6 without a preceding statement card
2	END IF before IF THEN or ELSE encountered
3	END WHILE before DO WHILE encountered
6	END CASE before CASE OF, CASE, or CASE ELSE encountered
7	END UNTIL before DO UNTIL encountered
9	ELSE after ELSE
11	CASE after CASE ELSE
12	CASE ELSE after CASE ELSE
13	ELSE before IF encountered
14	CASE before CASE OF encountered
15	Structural/syntactic error in program, cause unknown*
17	CASE ELSE before CASE OF or CASE encountered
19	Character string too long in CASE statement
100	Program END without balancing blocks for each statement
321	Improperly nested DMATRAN statements
999	Cause unknown. Unrecognizable statement.

*This occurs when the indentation level computed by STRUCTRAN-1 would become negative if the error was not detected. Possible causes are too many ELSE, END IF, END WHILE, END UNTIL, or END CASE statements.

4 STRUCTRAN-1 GUIDELINES

The following guidelines should be kept in mind when using STRUCTRAN-1:

1. A maximum of 20 cards per statement.

2. STRUCTRAN-1 generates FORTRAN GOTO statements and statement labels. Statement labels in the DMATRAN input source (FORMAT statements) should not duplicate the labels appearing in the translated FORTRAN.

3. When the DO UNTIL...END UNTIL construct is used for iteration, it is important to note that the statements contained within the construct will be executed once before the logical expression is evaluated.

4. All two word DMATRAN directives may be written as two separate words or merged into one; e.g., DOUNTIL or DO UNTIL.

5. A maximum of 20 BLOCKs per module.

6. A limit of 15 INVOKEs for any one BLOCK.

7. INVOKE for the BLOCK must occur before the BLOCK code (suggest BLOCKs be grouped at the end of the routine).

8. First six characters of the name of a BLOCK must be unique within a module.

9. Maximum nesting level for indentation of FORTRAN output is 10.

10. The value of <integer-expression> in CASE statements must be positive.

11. BLOCK constructs cannot contain labeled statements which are referred to outside of the BLOCK.

12. A BLOCK cannot be entered by falling into it (as the next executable statement).

APPENDIX A

STRUCTRAN-1 ON THE UNIVAC 1108

Various modifications to the STRUCTRAN-1 preprocessor have been made for ease of use in the UNIVAC 1108 environment. DMATRAN programs are input to the preprocessor by providing them as data in the run stream or by using the @ADD capability to add program library elements to the run stream. When used with a program library, the indented DMATRAN listing serves as the reference from which corrections to program library elements can be made. The use of DMATRAN with card input will be described first, followed by suggested procedures for use with a program library.

A.1 CARD INPUT

The STRUCTRAN-1 preprocessor accepts run stream data as input and outputs an indented source listing and invokes the FORTRAN V compiler to compile the translated FORTRAN code. Each and every input routine being processed by STRUCTRAN-1 must be proceeded by a DMATRAN control card containing the name desired for the corresponding FORTRAN symbolic element. A typical deck setup for compiling and executing DMATRAN source cards follows:

```
        Assign permanent file containing STRUCTRAN-1 and use PF1
        as its short name
@XQT PF1.STRUCTRAN1  (Executive card--blank if default parameters)
MAIN    (DMATRAN control card)
        DMATRAN source cards for main program
EXAMPL    (DMATRAN control card)
        DMATRAN source cards for subroutine EXAMPL
@XQT
        Data for MAIN and EXAMPL
@FIN
```

A DMATRAN control card is required as the first card of the DMATRAN input deck and after each FORTRAN END statement which is not the last statement of the input deck. The above example generates DMATRAN and FORTRAN source listings. Additionally, it creates FORTRAN elements MAIN and EXAMPL and relocatable elements MAIN and EXAMPL. It is recommended that the DMATRAN control cards provide element names which correspond to the subroutine names of the DMATRAN modules. Since the DMATRAN control card information is used for headings in the DMATRAN listing, this naming procedure assists in maintaining an alphabetically sorted listing of source routines.

A.2 PROGRAM LIBRARY

The @ADD capability allows larger DMATRAN programs to be conveniently implemented and maintained on a program library. Extensive capabilities of the ELT processor are then available for correcting/modifying the DMATRAN source code. The following run stream will create a program library for MAIN and EXAMPL:

```
      Assign permanent files
@ELT,I    MAIN/STR
MAIN      (DMATRAN control card)
      DMATRAN source text for main program
@ELT,I    EXAMPL/STR
EXAMPL    (DMATRAN control card)
      DMATRAN source text for subroutine EXAMPL
@XQT  PF1.STRUCTRAN1    (Execute STRUCTRAN-1)
@ADD  MAIN/STR
@ADD  EXAMPL/STR
@COPY  TPF$.,PF2.
@XQT      (Execute user's program)
      Data for MAIN and EXAMPL
@FIN
```

This run stream creates a program file in TPF$ containing DMATRAN source, FORTRAN source, and relocatable elements. The first card of each DMATRAN source element is a DMATRAN control card. FORTRAN and DMATRAN source elements cannot have the same element and version names and reside on the same program file. It is convenient to give all DMATRAN source elements a version name of STR to overcome this difficulty. The program library in TPF$ is saved by copying it to PF2. A DMATRAN listing is also produced by the above run stream. This listing should be saved so that it can be used to make modifications to the program library elements. The leftmost column of numbers on the listing are directly usable for making ELT modifications. This listing is more convenient to refer to than an ELT listing because it is indented.

NOTE: PF2. in the above example will contain the symbolic elements MAIN/STR and EXAMPL and the relocatable elements MAIN and EXAMPL.

Once a DMATRAN program library has been created and saved, the following run stream temporarily updates it and tests the updates.

```
      Assign permanent files
@COPY  PF2.,TPF$.
```

```
@ELT,U  MAIN/STR

        Main program corrections

@ELT,U  EXAMPL/STR

        Subroutine EXAMPL corrections

@XQT    PF1.STRUCTRAN1

@ADD    MAIN/STR

@ADD    A/STR

@XQT

        Data for MAIN and EXAMPL

@FIN
```

This run stream results in an updated DMATRAN program library in TPF$. After the updates have been found to be correct, the DMATRAN program file can be permanently updated by copying TPF$ to PF2. The DMATRAN listing of the corrected elements should be saved fot making further corrections.

INDEX

INDEX (Contd.)