USGS
science for a changing world

Techniques of Water-Resources Investigations
of the United States Geological Survey

Chapter A1

A MODULAR THREE-DIMENSIONAL FINITE-DIFFERENCE GROUND-WATER FLOW MODEL

By Michael G. McDonald and
Arlen W. Harbaugh

This chapter supersedes U.S. Geological Survey Open-File Report 83–875

Book 6

MODELING TECHNIQUES

Narrative for Module UCOLNO

Module UCOLNO prints column numbers at the top of a page when arrays of numbers are printed. It performs its functions in the following order:

1. Calculate the number of columns to be printed (NLBL), the width of a line (NTOT), and the number of lines needed to print all of the column numbers (NWRAP). Initialize the fields J1 and J2 which contain the first and last column number on each print line.

2. Build and print each line (DO STEPS 3-6).

3. Clear the line buffer (BF) in which the line is built.

4. Determine the first (J1) and last (J2) column number for the current line.

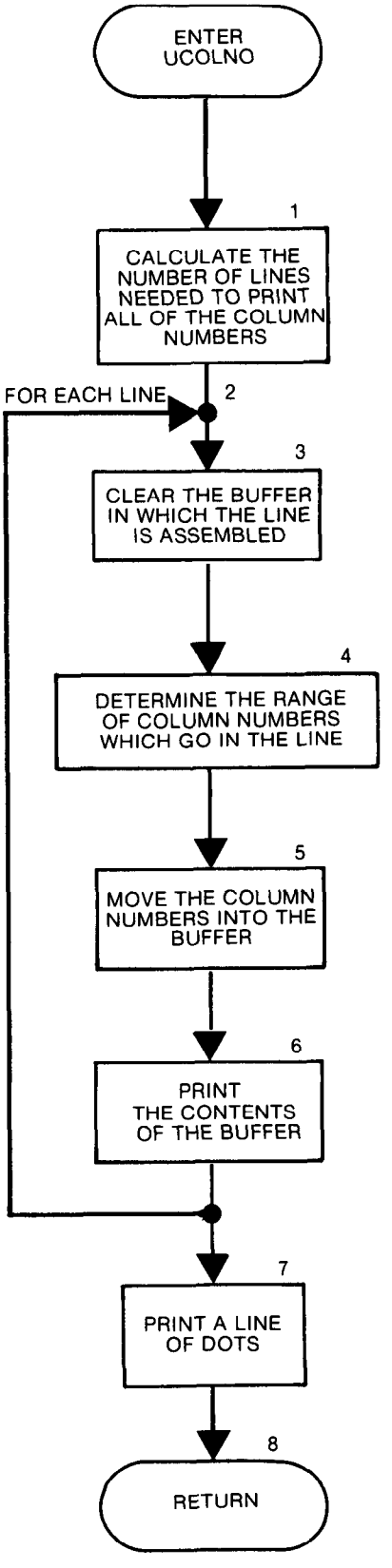
5. Put the column numbers in the line buffer. They are selected from the array DG. The indices I1, I2, and I3 point to the units digit, tens digit, and hundreds digit, respectively.

6. Print the line.

7. Print a line of dots.

8. RETURN.

Flow Chart for Module UCOLNO



```
      SUBROUTINE UCOLNO(NLBL1,NLBL2,NSPACE,NCPL,NDIG,IOUT)
C
C
C-----VERSION 1638 12MAY1987 UCOLNO
C     ******************************************************************
C     OUTPUT COLUMN NUMBERS ABOVE A MATRIX PRINTOUT
C        NLBL1 IS THE START COLUMN LABEL (NUMBER)
C        NLBL2 IS THE STOP COLUMN LABEL (NUMBER)
C        NSPACE IS NUMBER OF BLANK SPACES TO LEAVE AT START OF LINE
C        NCPL IS NUMBER OF COLUMN NUMBERS PER LINE
C        NDIG IS NUMBER OF CHARACTERS IN EACH COLUMN FIELD
C        IOUT IS OUTPUT CHANNEL
C     ******************************************************************
C
C        SPECIFICATIONS:
C     ------------------------------------------------------------------
      CHARACTER*4 DOT,SPACE,DG,BF
      DIMENSION BF(130),DG(10)
C
      DATA DG(1),DG(2),DG(3),DG(4),DG(5),DG(6),DG(7),DG(8),DG(9),DG(10)/
     1         '0   ','1   ','2   ','3   ','4   ','5   ','6   ',
     2         '7   ','8   ','9   '/
      DATA DOT,SPACE/'.   ','    '/
C     ------------------------------------------------------------------
C
C1------CALCULATE # OF COLUMNS TO BE PRINTED (NLBL), WIDTH
C1------OF A LINE (NTOT), NUMBER OF LINES (NWRAP).
      WRITE(IOUT,1)
    1 FORMAT(1X)
      NLBL=NLBL2-NLBL1+1
      N=NLBL
      IF(NLBL.GT.NCPL) N=NCPL
      NTOT=NSPACE+N*NDIG
      IF(NTOT.GT.130) GO TO 50
      NWRAP=(NLBL-1)/NCPL + 1
      J1=NLBL1-NCPL
      J2=NLBL1-1
C
C2------BUILD AND PRINT EACH LINE
      DO 40 N=1,NWRAP
C
C3------CLEAR THE BUFFER (BF).
      DO 20 I=1,130
      BF(I)=SPACE
   20 CONTINUE
      NBF=NSPACE
C
C4------DETERMINE FIRST (J1) AND LAST (J2) COLUMN # FOR THIS LINE.
      J1=J1+NCPL
      J2=J2+NCPL
      IF(J2.GT.NLBL2) J2=NLBL2
C5------LOAD THE COLUMN #'S INTO THE BUFFER.
      DO 30 J=J1,J2
      NBF=NBF+NDIG
      I2=J/10
      I1=J-I2*10+1
      BF(NBF)=DG(I1)
      IF(I2.EQ.0) GO TO 30
      I3=I2/10
      I2=I2-I3*10+1
      BF(NBF-1)=DG(I2)
      IF(I3.EQ.0) GO TO 30
      BF(NBF-2)=DG(I3+1)
   30 CONTINUE
C
C6------PRINT THE CONTENTS OF THE BUFFER (I.E. PRINT THE LINE).
      WRITE(IOUT,31) (BF(I),I=1,NBF)
   31 FORMAT(1X,130A1)
C
   40 CONTINUE
C
C7------PRINT A LINE OF DOTS (FOR ESTHETIC PURPOSES ONLY).
   50 NTOT=NTOT+5
      IF(NTOT.GT.130) NTOT=130
      WRITE(IOUT,51) (DOT,I=1,NTOT)
   51 FORMAT(1X,130A1)
C
C8------RETURN
      RETURN
      END
```

List of Variables for Module UCOLNO

Variable	Range	Definition
BF	Module	DIMENSION (130), Buffer in which a line is assembled.
DG	Module	DIMENSION (10), Digits 0 through 9.
DOT	Module	Field containing a period.
I	Module	Index for BF.
IOUT	Global	Primary unit number for all printed output. IOUT = 6.
I1	Module	Index for DG (units digit).
I2	Module	Index for DG (tens digit).
I3	Module	Index for DG (hundreds digit).
J	Module	Index for column numbers.
J1	Module	First column number on the current line.
J2	Module	Last column number on the current line.
N	Module	Number of column numbers.
NBF	Module	Index for BF.
NCPL	Module	Number of column numbers per line.
NDIG	Module	Number of characters in each column number field.
NLBL	Module	Number of column numbers to be printed.
NLBL1	Module	Start column number.
NLBL2	Module	Stop column number.
NSPACE	Module	Number of blank spaces at start of line.
NTOT	Module	Total number of characters on a line.
NWRAP	Module	Number of lines needed in wrap format.
SPACE	Module	Field containing blanks.

Narrative for Module U2DREL

Module U2DREL reads values for a two-dimensional real array. First it reads an "array-control record." Then, based on the contents of the array-control record, it may read array values. The array-control record contains four fields: location (LOCAT), constant (CNSTNT), format (FMTIN), and printout indicator (IPRN). The LOCAT field determines where array values will come from. If LOCAT is positive, it is the unit number from which array values will be read in the format specified in FMTIN. If LOCAT is negative, the sign is reversed to give the unit number from which an unformatted record containing the array values will be read. (Before the array record is read, a record will be read and ignored. Thus output from the module ULASAV can be read.) If LOCAT is zero, all of the array values will be set equal to CNSTNT. When LOCAT is not zero and CNSTNT is not zero, the array values will be multiplied by the value of CNSTNT. The field IPRN contains a code number for a FORTRAN format to be used when printing the array.

Module U2DREL performs its tasks in the following order:

1. Read the array-control record (LOCAT, CNSTNT, FMTIN, and IPRN).

2. Use LOCAT to determine where the array values are coming from. GO TO STEPS 3, 4, OR 5.

3. If LOCAT equals zero, set all array values equal to CNSTNT, print a message to that effect, and RETURN.

4. If LOCAT is greater than zero, read array values according to the format in FMTIN. GO TO STEP 6.

5. If LOCAT is less than zero, read an unformatted dummy record and then read an unformatted record containing the array values. GO TO STEP 6.

6. If CNSTNT is not equal to zero, multiply array values by CNSTNT.

7. If IPRN is greater than or equal to zero, call utility module ULAPRW to print the array values using IPRN as the format code.

8. RETURN.

Flow Chart for Module U2DREL

Array Control Record controls the input of array values. It contains four fields: LOCAT, CNSTNT, FMTIN, and IPRN.

LOCAT is a code showing where array values will come from.

If LOCAT < 0, array values will be read from an unformatted record from a unit number equal to -LOCAT.

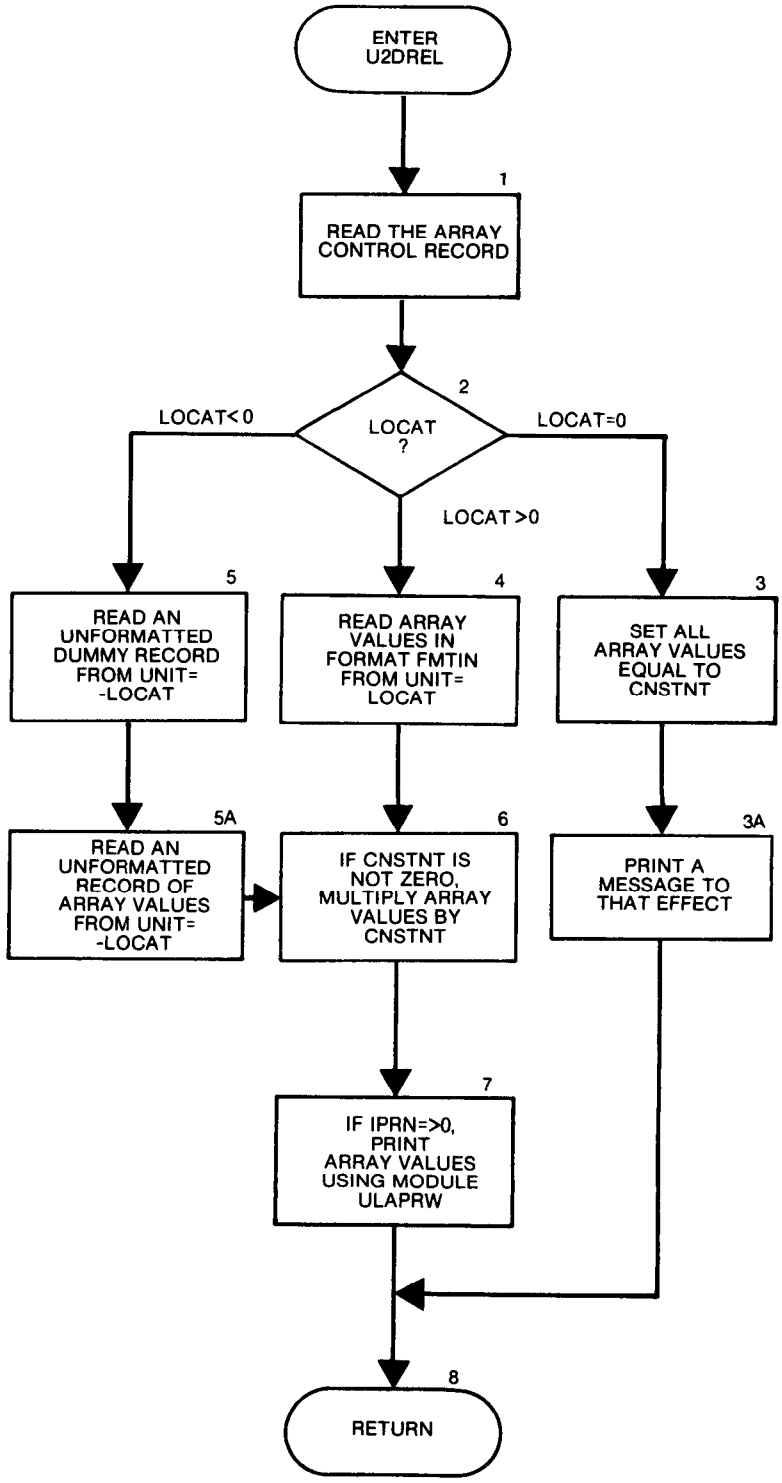
If LOCAT = 0, array values will be set equal to CNSTNT.

If LOCAT > 0, array values will be read from the unit number equal to LOCAT in the format specified in FMTIN.

CNSTNT is the constant to which all array values are set if LOCAT is equal to zero, and it is the constant by which all array values are multiplied if LOCAT is not equal to zero.

FMTIN is the format in which array values are read if LOCAT is greater than zero.

IPRN is a code showing the format to be used if array values are to be printed.



```
      SUBROUTINE U2DREL(A,ANAME,II,JJ,K,IN,IOUT)
C
C
C-----VERSION 1648 12MAY1987 U2DREL
C     *****************************************************************
C     ROUTINE TO INPUT 2-D REAL DATA MATRICES
C       A IS ARRAY TO INPUT
C       ANAME IS 24 CHARACTER DESCRIPTION OF A
C       II IS NO. OF ROWS
C       JJ IS NO. OF COLS
C       K IS LAYER NO. (USED WITH NAME TO TITLE PRINTOUT UNLESS K IS 0)
C       IN IS INPUT UNIT
C       IOUT IS OUTPUT UNIT
C     *****************************************************************
C
C        SPECIFICATIONS:
C     -----------------------------------------------------------------
      CHARACTER*4 ANAME
      CHARACTER*20 FMTIN
      DIMENSION A(JJ,II),ANAME(6)
C     -----------------------------------------------------------------
C
C1------READ ARRAY CONTROL RECORD.
      READ (IN,1) LOCAT,CNSTNT,FMTIN,IPRN
    1 FORMAT(I10,F10.0,A20,I10)
C
C2------USE LOCAT TO SEE WHERE ARRAY VALUES COME FROM.
      IF(LOCAT) 200,50,90
C
C3------IF LOCAT=0 THEN SET ALL ARRAY VALUES EQUAL TO CNSTNT. RETURN
   50 DO 80 I=1,II
      DO 80 J=1,JJ
   80 A(J,I)=CNSTNT
      IF(K.GT.0) WRITE(IOUT,2) ANAME,CNSTNT,K
    2 FORMAT(1H0,52X,6A4,' =',G15.7,' FOR LAYER',I3)
      IF(K.LE.0) WRITE(IOUT,3) ANAME,CNSTNT
    3 FORMAT(1H0,52X,6A4,' =',G15.7)
      RETURN
C
C4------IF LOCAT>0 THEN READ FORMATTED RECORDS USING FORMAT FMTIN.
   90 IF(K.GT.0) WRITE(IOUT,4) ANAME,K,LOCAT,FMTIN
    4 FORMAT(1H0,///30X,6A4,' FOR LAYER',I3,' WILL BE READ ON UNIT',
     1       I3,' USING FORMAT: ',A20/30X,96('-'))
      IF(K.LE.0) WRITE(IOUT,5) ANAME,LOCAT,FMTIN
    5 FORMAT(1H0,///30X,6A4,' WILL BE READ ON UNIT',
     1       I3,' USING FORMAT: ',A20/30X,83('-'))
      DO 100 I=1,II
      READ (LOCAT,FMTIN) (A(J,I),J=1,JJ)
  100 CONTINUE
      GO TO 300
C
C5------LOCAT<0 THEN READ UNFORMATTED RECORD CONTAINING ARRAY VALUES
  200 LOCAT=-LOCAT
      IF(K.GT.0) WRITE(IOUT,201) ANAME,K,LOCAT
  201 FORMAT(1H0,///30X,6A4,', LAYER',I3,
     1    ' WILL BE READ UNFORMATTED ON UNIT',I3/30X,73('-'))
      IF(K.LE.0) WRITE(IOUT,202) ANAME,LOCAT
  202 FORMAT(1H0,///30X,
     1    ' WILL BE READ UNFORMATTED ON UNIT',I3/30X,60('-'))
C
C5A-----READ AN UNFORMATTED DUMMY RECORD FIRST.
      READ(LOCAT)
      READ(LOCAT) A
C
C6------IF CNSTNT NOT ZERO THEN MULTIPLY ARRAY VALUES BY CNSTNT.
  300 IF(CNSTNT.EQ.0.) GO TO 320
      DO 310 I=1,II
      DO 310 J=1,JJ
      A(J,I)=A(J,I)*CNSTNT
  310 CONTINUE
C
C7------IF PRINT CODE (IPRN) =>0 THEN PRINT ARRAY VALUES.
  320 IF(IPRN.LT.0) RETURN
      CALL ULAPRW(A,ANAME,0,0,JJ,II,0,IPRN,IOUT)
      RETURN
C
C8------RETURN
      END
```

List of Variables for Module U2DREL

Variable	Range	Definition
A	Module	DIMENSION (JJ,II), Input array.
ANAME	Module	Label for printout of the input array.
CNSTNT	Module	Constant to which all array values are set if LOCAT is equal to zero or by which all array values are multiplied if LOCAT is not equal to zero.
FMTIN	Module	Format under which array values will be read.
I	Module	Index for rows.
II	Module	Number of rows.
IN	Module	Unit number from which the array control record will be read.
IOUT	Global	Primary unit number for all printed output. IOUT = 6.
IPRN	Module	Code for format to be used when printing the arrays.
J	Module	Index for columns.
JJ	Module	Number of columns.
K	Module	Layer number.
LOCAT	Module	Location of values to fill in the array. < 0, read an unformatted record containing the array values. = 0, set all the array values equal to constant (CNSTNT). > 0, read the formatted records containing the array values.

Narrative for Module U2DINT

Module U2DINT reads values for a two-dimensional integer array. First it reads an "array-control record." Then, based on the contents of the array-control record, it may read array values. The array-control record contains four fields: location (LOCAT), constant (ICONST), format (FMTIN), and printout indicator (IPRN). The LOCAT field determines where array values will come from. If LOCAT is positive, it is the unit number from which array values will be read in the format specified in FMTIN. If LOCAT is negative, the sign is reversed to give the unit number from which an unformatted record containing the array values will be read. (Before the array record is read, a record will be read and ignored. Thus output from the module ULASAV can be read.) If LOCAT is zero, all of the array values will be set equal to ICONST. When LOCAT is not zero and ICONST is not zero, the array values will be multiplied by the value of ICONST. The field IPRN (table 1) contains a code number for a FORTRAN format to be used when printing the array.

Module U2DINT performs its tasks in the following order:

1. Read the array-control record (LOCAT, ICONST, FMTIN, and IPRN).

2. Use LOCAT to determine where the array is coming from. GO TO STEPS 3, 4, OR 5.

3. If LOCAT equals zero, set all array values equal to CNSTNT, print a message to that effect, and RETURN.

4. If LOCAT is greater than zero, read array values according to the format in FMTIN. GO TO STEP 6.

5. If LOCAT is less than zero, read an unformatted dummy record and then read an unformatted record containing the array values. GO TO STEP 6.

6. If ICONST is not equal to zero, multiply array values by ICONST.

7. If IPRN is greater than or equal to zero, print the array values using IPRN as the format code.

8. Call utility module UCOLNO to print column numbers at the top of the page.

9. Print each row in the array.

10. Select the format for printing.

11. RETURN.

Flow Chart for Module U2DINT

Array Control Record controls the input of array values. It contains four fields: LOCAT, ICONST, FMTIN, and IPRN.

LOCAT is a code showing where array values will come from.

If LOCAT < 0, array values will be read from an unformatted record from a unit number equal to -LOCAT.

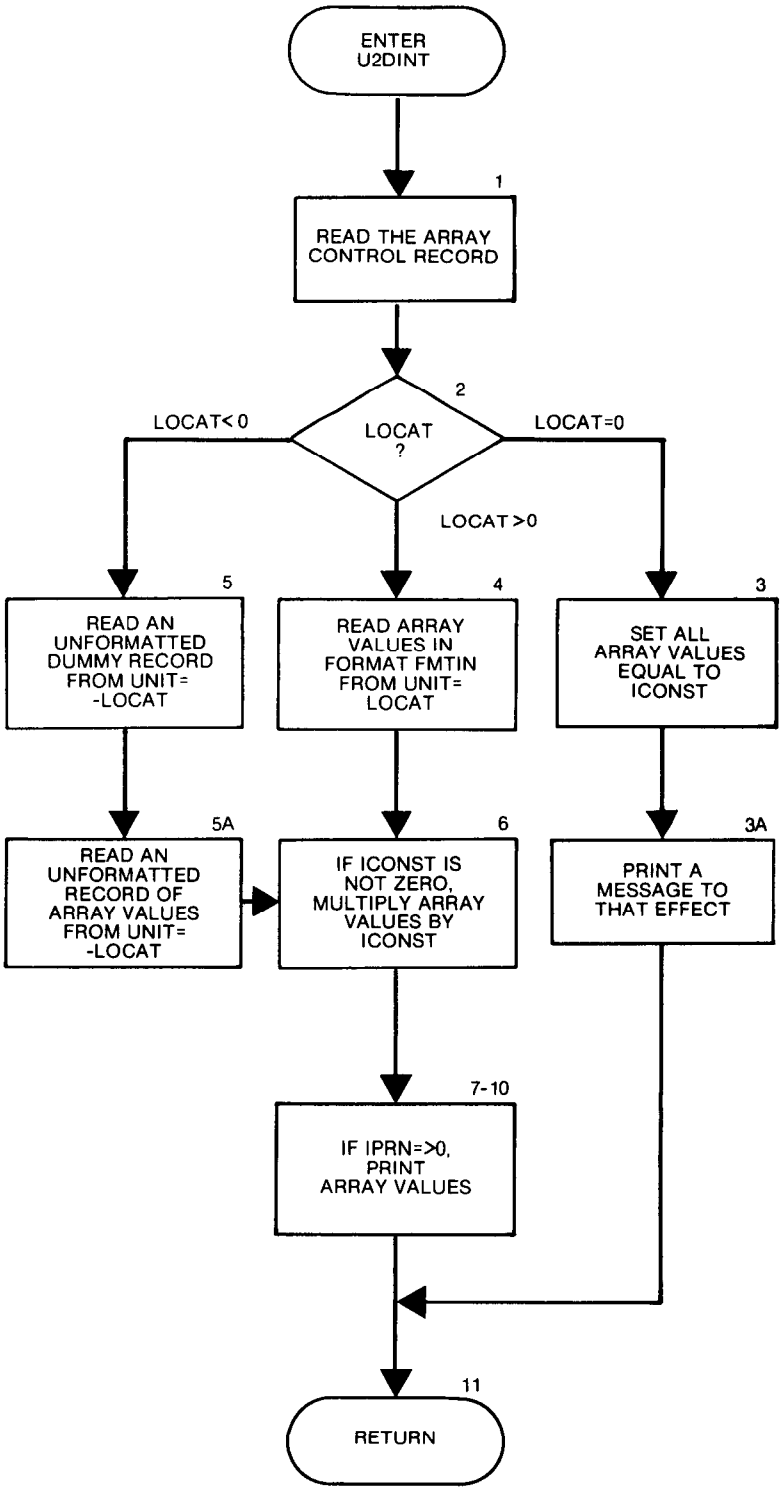
If LOCAT = 0, array values will be set equal to ICONST.

If LOCAT > 0, array values will be read from the unit number equal to LOCAT in the format specified in FMTIN.

ICONST is the constant to which all array values are set if LOCAT is equal to zero, and it is the constant by which all array values are multiplied if LOCAT is not equal to zero.

FMTIN is the format in which array values are read if LOCAT is greater than zero.

IPRN is a code showing the format to be used if array values are to be printed.



```
      SUBROUTINE U2DINT(IA,ANAME,II,JJ,K,IN,IOUT)
C
C
C-----VERSION 1645 12MAY1987 U2DINT
C     ******************************************************************
C     ROUTINE TO INPUT 2-D INTEGER DATA MATRICES
C       IA IS ARRAY TO INPUT
C       ANAME IS 24 CHARACTER DESCRIPTION OF IA
C       II IS NO. OF ROWS
C       JJ IS NO. OF COLS
C       K IS LAYER NO. (USED WITH NAME TO TITLE PRINTOUT UNLESS K IS 0)
C       IN IS INPUT UNIT
C       IOUT IS OUTPUT UNIT
C     ******************************************************************
C
C        SPECIFICATIONS:
C     ------------------------------------------------------------------
      CHARACTER*4 ANAME
      CHARACTER*20 FMTIN
      DIMENSION IA(JJ,II),ANAME(6)
C     ------------------------------------------------------------------
C
C1------READ ARRAY CONTROL RECORD.
      READ (IN,1) LOCAT,ICONST,FMTIN,IPRN
    1 FORMAT(I10,I10,A20,I10)
C
C2------USE LOCAT TO SEE WHERE ARRAY VALUES COME FROM.
      IF(LOCAT) 200,50,90
C
C3------IF LOCAT=0 THEN SET ALL ARRAY VALUES EQUAL TO ICONST. RETURN
   50 DO 80 I=1,II
      DO 80 J=1,JJ
   80 IA(J,I)=ICONST
      IF(K.GT.0) WRITE(IOUT,2) ANAME,ICONST,K
    2 FORMAT(1H0,52X,6A4,' =',I15,' FOR LAYER',I3)
      IF(K.LE.0) WRITE(IOUT,3) ANAME,ICONST
    3 FORMAT(1H0,52X,6A4,' =',I15)
      RETURN
C
C4------IF LOCAT>0 THEN READ FORMATTED RECORDS USING FORMAT FMTIN.
   90 IF(K.GT.0) WRITE(IOUT,4) ANAME,K,LOCAT,FMTIN
    4 FORMAT(1H0,///30X,6A4,' FOR LAYER',I3,' WILL BE READ ON UNIT',
     1       I3,' USING FORMAT: ',A20/30X,96('-'))
      IF(K.LE.0) WRITE(IOUT,5) ANAME,LOCAT,FMTIN
    5 FORMAT(1H0,///30X,6A4,' WILL BE READ ON UNIT',
     1        I3,' USING FORMAT: ',A20/30X,83('-'))
      DO 100 I=1,II
      READ (LOCAT,FMTIN) (IA(J,I),J=1,JJ)
  100 CONTINUE
      GO TO 300
C
C5------LOCAT<0 THEN READ UNFORMATTED RECORD CONTAINING ARRAY VALUES
  200 LOCAT=-LOCAT
      IF(K.GT.0) WRITE(IOUT,201) ANAME,K,LOCAT
  201 FORMAT(1H0,///30X,6A4,', LAYER',I3,
     1    ' WILL BE READ UNFORMATTED ON UNIT',I3/30X,73('-'))
      IF(K.LE.0) WRITE(IOUT,202) ANAME,LOCAT
  202 FORMAT(1H0,///30X,6A4,
     1    ' WILL BE READ UNFORMATTED ON UNIT',I3/30X,60('-'))
C
```

```
C5A------READ AN UNFORMATTED DUMMY RECORD FIRST.
      READ(LOCAT)
      READ(LOCAT) IA
C
C6------IF ICONST NOT ZERO THEN MULTIPLY ARRAY VALUES BY ICONST.
  300 IF(ICONST.EQ.0) GO TO 320
      DO 310 I=1,II
      DO 310 J=1,JJ
      IA(J,I)=IA(J,I)*ICONST
  310 CONTINUE
C
C7------IF PRINT CODE (IPRN) =>0 THEN PRINT ARRAY VALUES.
  320 IF(IPRN.LT.0) RETURN
      IF(IPRN.GT.5) IPRN=0
      IPRN=IPRN+1
C
C8------PRINT COLUMN NUMBERS AT TOP OF PAGE.
      IF(IPRN.EQ.1) CALL UCOLNO(1,JJ,0,10,12,IOUT)
      NL=125/IPRN/5*5
      IF(IPRN.GT.1) CALL UCOLNO(1,JJ,4,NL,IPRN,IOUT)
C
C9------PRINT EACH ROW IN THE ARRAY.
      DO 110 I=1,II
C
C10-----SELECT THE FORMAT
      GO TO(101,102,103,104,105,106), IPRN
C
C----------------FORMAT 10I11
  101 WRITE(IOUT,1001) I,(IA(J,I),J=1,JJ)
 1001 FORMAT(1H0,I3,2X,I11,9(1X,I11)/(5X,10(1X,I11)))
      GO TO 110
C
C----------------FORMAT 60I1
  102 WRITE(IOUT,1002) I,(IA(J,I),J=1,JJ)
 1002 FORMAT(1H0,I3,1X,60(1X,I1)/(5X,60(1X,I1)))
      GO TO 110
C
C----------------FORMAT 40I2
  103 WRITE(IOUT,1003) I,(IA(J,I),J=1,JJ)
 1003 FORMAT(1H0,I3,1X,40(1X,I2)/(5X,40(1X,I2)))
      GO TO 110
C
C----------------FORMAT 30I3
  104 WRITE(IOUT,1004) I,(IA(J,I),J=1,JJ)
 1004 FORMAT(1H0,I3,1X,30(1X,I3)/(5X,30(1X,I3)))
      GO TO 110
C
C----------------FORMAT 25I4
  105 WRITE(IOUT,1005) I,(IA(J,I),J=1,JJ)
 1005 FORMAT(1H0,I3,1X,25(1X,I4)/(5X,25(1X,I4)))
      GO TO 110
C
C----------------FORMAT 20I5
  106 WRITE(IOUT,1006) I,(IA(J,I),J=1,JJ)
 1006 FORMAT(1H0,I3,1X,20(1X,I5)/(5X,20(1X,I5)))
  110 CONTINUE
      RETURN
C
C11-----RETURN
      END
```

List of Variables for Module U2DINT

Variable	Range	Definition
ANAME	Module	Label for the printout of input array.
FMTIN	Module	Format under which the array values will be read.
I	Module	Index for rows.
IA	Module	DIMENSION (JJ,II), Input array.
ICONST	Module	Constant to which all array values are set if LOCAT is equal to zero or by which all array values are multiplied if LOCAT is not equal to zero.
II	Module	Number of rows.
IN	Module	Unit number from which the array-control record will be read.
IOUT	Global	Primary unit number for all printed output. IOUT = 6.
IPRN	Module	Code for format to be used when printing arrays.
J	Module	Index for columns.
JJ	Module	Number of columns.
K	Module	Layer number.
LOCAT	Module	Location of values to fill in the array. < 0, read an unformatted record containing the array values. = 0, set all the array values equal to constant (CNSTNT). > 0, read formatted records containing the array values.
NL	Module	Number of columns per line.

Narrative for Module U1DREL

Module U1DREL reads values for a one-dimensional real array. First it reads an "array-control record." Then, based on the contents of the array-control record, it may read array values. The array-control record contains four fields: location (LOCAT), constant (CNSTNT), format (FMTIN), and printout indicator (IPRN). The LOCAT field determines where array values will come from. If LOCAT is positive, it is the unit number from which array values will be read in the format specified in FMTIN. If LOCAT is zero, all of the array values will be set equal to CNSTNT. If LOCAT is not zero and CNSTNT is not zero, the array values will be multiplied by the value of CNSTNT. The field IPRN (table 2) contains a code number for a FORTRAN format to be used when printing the array.

Module U1DREL performs its tasks in the following order:

1. Read the array-control record (LOCAT, CNSTNT, FMTIN, and IPRN).

2. Use LOCAT to determine where the array is coming from (DO STEPS 3 OR 4).

3. If LOCAT equals zero, set all array values equal to CNSTNT and print a message to that effect. RETURN.

4. If LOCAT is greater than zero, read array values according to the format in FMTIN.

5. If CNSTNT is not equal to zero, multiply the array values by CNSTNT.

6. If IPRN is greater than or equal to zero, print the array values.

7. RETURN.

Flow Chart for Module U1DREL

Array Control Record controls the input of array values. It contains four fields: LOCAT, CNSTNT, FMTIN, and IPRN.

LOCAT is a code showing where array values will come from.

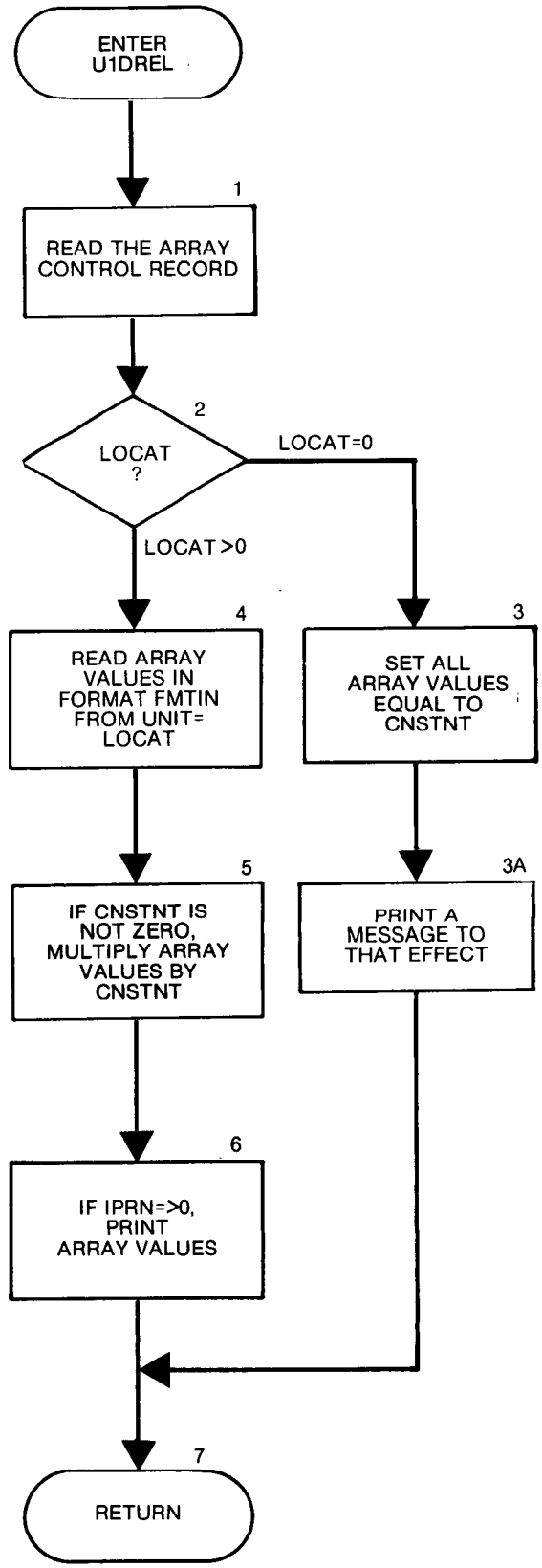
If LOCAT = 0, array values will be set equal to CNSTNT.

If LOCAT > 0, array values will be read from the unit number equal to LOCAT in the format specified in FMTIN.

CNSTNT is the constant to which all array values are set if LOCAT is equal to zero, and it is the constant by which all array values are multiplied if LOCAT is not equal to zero.

FMTIN is the format in which array values are read if LOCAT is greater than zero.

IPRN is a code showing the format to be used if array values are to be printed.



```
      SUBROUTINE U1DREL(A,ANAME,JJ,IN,IOUT)
C
C
C-----VERSION 1643 12MAY1987 U1DREL
C     ******************************************************************
C     ROUTINE TO INPUT 1-D REAL DATA MATRICES
C       A IS ARRAY TO INPUT
C       ANAME IS 24 CHARACTER DESCRIPTION OF A
C       JJ IS NO. OF ELEMENTS
C       IN IS INPUT UNIT
C       IOUT IS OUTPUT UNIT
C     ******************************************************************
C
C        SPECIFICATIONS:
C     ------------------------------------------------------------------
      CHARACTER*4 ANAME
      CHARACTER*20 FMTIN
      DIMENSION A(JJ),ANAME(6)
C     ------------------------------------------------------------------
C
C1------READ ARRAY CONTROL RECORD.
      READ (IN,1) LOCAT,CNSTNT,FMTIN,IPRN
    1 FORMAT(I10,F10.0,A20,I10)
C
C2------USE LOCAT TO SEE WHERE ARRAY VALUES COME FROM.
      IF(LOCAT.GT.0) GO TO 90
C
C3------IF LOCAT=0 THEN SET ALL ARRAY VALUES EQUAL TO CNSTNT. RETURN
      DO 80 J=1,JJ
   80 A(J)=CNSTNT
      WRITE(IOUT,3) ANAME,CNSTNT
    3 FORMAT(1H0,52X,6A4,' =',G15.7)
      RETURN
C
C4------IF LOCAT>0 THEN READ FORMATTED RECORDS USING FORMAT FMTIN.
   90 WRITE(IOUT,5) ANAME,LOCAT,FMTIN
    5 FORMAT(1H0,///30X,6A4,' WILL BE READ ON UNIT',I3,
     1         ' USING FORMAT: ',A20/30X,79('-')/)
      READ (LOCAT,FMTIN) (A(J),J=1,JJ)
C
C5------IF CNSTNT NOT ZERO THEN MULTIPLY ARRAY VALUES BY CNSTNT.
      IF(CNSTNT.EQ.0.) GO TO 120
      DO 100 J=1,JJ
  100 A(J)=A(J)*CNSTNT
C
C6------IF PRINT CODE (IPRN) =>0 THEN PRINT ARRAY VALUES.
 120  IF(IPRN.LT.0) RETURN
      WRITE(IOUT,1001) (A(J),J=1,JJ)
 1001 FORMAT((1X,1PG12.5,9(1X,G12.5)))
      RETURN
C
C7------CONTINUE
      END
```

List of Variables for Module U1DREL

Variable	Range	Definition
A	Module	DIMENSION (JJ), Input array.
ANAME	Module	Label for printout of the input array.
CNSTNT	Module	Constant to which all array values are set if LOCAT is equal to zero or by which all array values are multiplied if LOCAT is not equal to zero.
FMTIN	Module	Format under which the array values will be read.
IN	Module	Unit number from which the array control record will be read.
IOUT	Global	Primary unit number for all printed output. IOUT = 6.
IPRN	Module	Code for the format to be used when printing the arrays.
J	Module	Array index.
JJ	Module	Number of elements in the array.
LOCAT	Module	Location of values to fill in the array. < 0, read an unformatted record containing the array values. = 0, set all the array values equal to constant (CNSTNT). > 0, read the formatted records containing the array values.

REFERENCES

Collins, R. E., 1961, Flow of fluids through porous materials; New York, Reinhold Publishing Corp., 270 p.

Crichlow, Henry B., 1977, Modern reservoir engineering - A simulation approach; Englewood Cliffs, N.J., Prentice Hall Inc., 354 p.

McDonald, M.G., and Harbaugh, A.W., 1984, A modular three-dimensional finite-difference ground-water flow model: U.S. Geological Survey Open-File Report 83-875 , 528 p.

Peaceman, Donald W., 1977, Fundamentals of numerical reservoir simulation; New York, Elsevier Scientific Publishing Company, 176 p.

Remson, Irwin, Hornberger, George M. and Molz, Fred J., 1971, Numerical methods in subsurface hydrology; New York, Wiley-Interscience, 389 p.

Rushton, K. R., Redshaw S. C., 1979, Seepage and groundwater flow-numerical analysis by analog and digital methods; New York, John Wiley and Sons

Trescott, Peter C., 1975, Documentation of finite-difference model for simulation of three-dimensional ground-water flow: U.S. Geological Survey Open-File Report 75-438, 32 p.

Trescott, Peter C. and Larson, S. P., 1976, Supplement to Open-File Report 75-438, Documentation of finite-difference model of three-dimensional ground-water flow; U.S. Geological Survey Open-File Report 76-591, 21 p.

Trescott, P. C., Pinder, G. F., and Larson, S. P., 1976, Finite-difference model for aquifer simulation in two dimensions with results of numerical experiments: U.S. Geological Survey Techniques of Water-Resources Investigations, Book 7, Chapter C1, 116 p.

Weinstein, H. C., Stone, H. L., and Kwan, T. V., 1969, Iterative procedure for solution of systems of parabolic and elliptic equations in three dimensions: Indus. Engineering Chemistry Fundamentals, v. 8, no. 2, p. 281-287.

APPENDIX A

PROGRAM PORTABILITY

Introduction

One of the major design requirements for the model program was that it should be portable. A portable program is one that can be run with a minimum of modification on most computers that are physically capable of running a program of its type. The goal of portability for the model program has been attained as evidenced by the fact that it has run successfully in either its present form or the earlier form (McDonald and Harbaugh, 1984) on computers manufactured by many companies including mainframe computers, minicomputers, and microcomputers. The following discussion explains in more detail the concept of portability, what was done to maximize portability of the model program, and circumstances that might require that the program be changed in order to run successfully on a particular computer.

The Impact of the Programming Language on Portability

The programming language is the most important factor that determines program portability. There are a variety of programming languages available, and for each language, there are numerous versions which have resulted from the desire of vendors to improve the power of the language and to take advantage of the hardware features of their particular computers. The most commonly available language suitable to use for the model program is FORTRAN. There are two versions defined by the American National Standards Institute (ANSI) on which most commercial versions are based, ANSI X3.9-1966 and ANSI X3.9-1978.[1] These versions are commonly referred to as FORTRAN 66 and FORTRAN 77, respectively. FORTRAN 66 was selected for the original version of the model because, at the time, it was far more widely supported than was FORTRAN 77. Now that the FORTRAN 77 standard is more widely supported, the program has been converted to this standard.

The program in this report is nearly identical to the original program except for the changes required to make it comply with the FORTRAN 77 standard. These changes are minor and are described by McDonald and Harbaugh (1984, p. 505). The conversion was done solely for the purpose of making the program comply with the FORTRAN 77 standard; many of the features that make FORTRAN 77 more powerful than FORTRAN 66 were not used. An effort to make more extensive FORTRAN 77 revisions to the program is judged to be uneconomical. Such an effort would not result in significant improvements in program clarity or efficiency.

---

[1] American National Standards Institute, 1966, FORTRAN: American National Standards Institute, X3.9-1966, 36 p.

American National Standards Institute, 1978, Programming language FORTRAN: American National Standards Institute, X3.9-1978, chs. 1-18.

Most commercial versions of FORTRAN 77 compilers include some extended features not defined as part of the FORTRAN 77 standard. Such features vary widely among computer vendors and were not used in the model program. The program contains only one exception to complete compliance to the FORTRAN 77 standard that the authors are aware of. This exception, which is explained in a following section (see The Impact of Allocating Array Storage in a Single Array on Portability), is commonly allowed on computers and should not be a major restriction to portability. Because the program closely follows the FORTRAN 77 standard, the program should work on any computer supporting this language provided that the computer has adequate computational power.

It is recognized that some users may want the program converted back into the FORTRAN 66 language because they have access only to older compilers. Only a few changes are required to convert the program back to FORTRAN 66. Information about how to convert is provided in a following section (see Conversion to FORTRAN 66).

The Impact of Computational Precision on Portability

Variation of precision among computers causes some problems with program portability. Computational precision refers to the accuracy at which numbers are calculated and stored in the computer. To prevent the imposition of constraints on the computers on which FORTRAN is implemented, the computational precision was not defined as part of the standards. The accuracy of model results are dependent on the computational precision, so precision must be considered when moving the model program among computers.

The model program was developed on computers using 32 binary bits to represent single precision real numbers. This gives from 6 to 7 decimal digits of precision and includes values that range in magnitude from approximately 10**-39 to 10**38. Double precision real numbers are represented by 64 binary bits and range in magnitude from approximately 10**-10000 to 10**10000 with 14 to 15 decimal digits of precision. The head array, HNEW, and some variables in the solvers are stored as double precision, and accordingly many calculations in the solvers are double precision. This was necessary for accuracy under some conditions. The model program should perform adequately for most problems on computers that use 32 bits for single precision and 64 bits for double precision. However, the required precision depends on the problem being simulated. Thus, the user must ultimately determine if adequate precision is being used for solving a particular problem.

There are some situations for which there is a need to modify the program to make all real number calculations in double precision. If using a computer that represents single precision real numbers with less than 32 bits, then double precision is probably necessary for all real numbers and calculations. Even on computers that represent single precision numbers with 32 bits, certain problems are difficult to solve without making all real numbers and calculations double precision. Unfortunately, it is difficult to predict if a specific problem requires all double precision. Simulations with very large numbers of cells, for example more than 50000, are more likely to have precision problems than are smaller simulations. Simulations in which there are areas having significant ground-water flow and yet the heads in the adjacent model cells in these areas are equal within .01 percent or less are

also more likely to have precision problems. In addition, precision problems depend on the preciseness of the attempted solution. In general, the symptoms of inadequate precision are either a poor volumetric flow balance or lack of convergence by the solver. However, these same symptoms are more commonly caused by bad input data or improper adjustment of parameters that control iteration. Because precision problems are in general fairly unlikely and use of all double precision results in increased memory and computer time usage, conversion to double precision should probably be done as a last resort in order to solve convergence problems. If the program is converted to all double precision, almost twice the computer memory is required for model data. To convert all real numbers and calculations to double precision, do the following:

1. Declare all single precision real arrays and variables as DOUBLE PRECISION in the main program and in every subroutine. This can be done by adding the statement

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
```

   to the beginning of the specifications section of every subroutine and the main program. Alternately, some compilers have a special command that will accomplish this without the need to modify the program itself.

2. In the Basic Package Allocate module (subroutine BAS1AL) change the statement

```
ISUM=ISUM+2*NRCL
```

   to

```
ISUM=ISUM+NRCL
```

3. Change the real intrinsic functions in subroutine SSIP1I to their double precision equivalents. Specifically, change all occurrences of AMAX1 to DMAX1, and AMIN1 to DMIN1.

If using a computer on which significantly more than 32 bits are used to represent single precision real numbers, then the partial use of double precision data and calculations as included in the present model program may be unnecessary. The program should still work on such computers without modification, but computational time and memory would be saved if double precision were eliminated. The program was designed so that changing to all single precision would be easy. For this reason, the use of double precision intrinsic functions and constants is avoided, and all conversions between real and double precision are done by implied type changes in assignment statements. If it is determined that the use of all single precision is acceptable, make the following changes to the program:

1. Delete all DOUBLE PRECISION specification statements.

2. In the Basic Package Allocate module (subroutine BAS1AL) change the statement

```
ISUM=ISUM+2*NRCL
```

   to

```
ISUM=ISUM+NRCL
```

The model program was developed on computers that use 32 bits for integer numbers and calculations. This is more than adequate to represent the range of integer numbers that are used in the program under any conditions. The largest integer that the computer must be able to represent is the dimensioned size of the X array in the main program. Many FORTRAN compilers allow one to specify that integers be represented by 16 bits, which does not provide enough precision for larger model simulations. That is, a 16 bit integer in a typical computer can represent numbers in the range -32768 to 32767, and an X array dimension of 32767 is adequate only for simulations that have 2200 to 3000 cells, depending on what options are used.

The Impact of Allocating Array Storage in a Single Array on Portability

Because almost all model data are stored in the X array, which is dimensioned in the main program, this array can be quite large. It is generally 10 to 15 times the size of the number of model cells. Some computers require that special compiler options be used when an array exceeds a specified size. Users are cautioned to be aware of this and use the appropriate options as needed. The result of using the incorrect option can sometimes be that the program will execute without producing error messages, but answers will be incorrect.

All computers limit the amount of array space that a program can use. If a user's simulation exceeds this limit, the only options may be to use a different computer or make the simulation smaller. However, some computers make a distinction between total amount of memory used for all arrays and the total used by a single array. That is, some computers might allow 4 arrays each having 32000 elements, but not allow a single array consisting of 128000 elements. On such a computer, it is quite possible that one could exceed the size limit for a single array without exceeding the total array size limit because nearly all model data are stored in the single X array. If this situation occurs, it is possible to break the X array into smaller pieces. Although this can be done by modifying only the main program, the modification is fairly complex. Such modification requires a knowledgeable programmer who has a clear understanding of how model data are stored within the X array. Before making such a modification, it would be prudent to assess how long the desired simulation might take to execute. Generally, computer execution speed is more of a constraint on maximum problem size than is array size.

The only known exception to the use of the FORTRAN 77 standard in the model program is that the data type of actual subroutine arguments does not always match the type of the corresponding dummy arguments. The standard requires the data type of actual and dummy arguments to match. This only happens in the main program where it calls subroutines using actual arguments that are elements from the X array. The X array itself is data type real, and several model arrays stored withi X are either integer (arrays IOFLG, IBOUND, IRCH, IEVT, and LRCH) or double precision (array HNEW). For example, actual argument X(LCIRCH) in the main program is passed to dummy argument IRCH in subroutine RCH1RP. X(LCIRCH) is of type real, and IRCH is of type integer. This practice has not been a problem in the past. Should this become a problem on future computers, required changes will be fairly minor, affecting only the main program and some of the Allocate modules.

The Impact on Portability of Preconnected File Units

FORTRAN 77 provides 2 ways for a file unit to become connected to (associated with) a file -- preconnection and the OPEN statement. The model program use preconnection. This means that the computer's operating system provides the connection between files and file units prior to program execution. Often the user must issue commands to the system, which connect the necessary files and file units, prior to running the program. The specific method for doing this varies among computers. Generally, preconnection is adequate, but it may be inconvenient on some computers. Modification of the main program to use OPEN statements is a simple task for a programmer.

Conversion to FORTRAN 66

Because the model program uses only one feature of FORTRAN 77 that is not part of FORTRAN 66, few changes are needed in order to make the program comply with the FORTRAN 66 standard. All of the required changes are a result of differences in the way character (alphanumeric) data are handled by the two versions of FORTRAN. Any variables or arrays holding character data must be declared to be the character data type in a FORTRAN 77 program; in a FORTRAN 66 program, character data are stored in numeric variables or arrays. The specific changes that are required to make the model program comply with the FORTRAN 66 standard are shown below. It is assumed that at least four characters can be stored in a single precision real variable or array element.

1. Delete all CHARACTER*4 statements throughout all subroutines and the main program.

2. In each of the array reading utility modules (U1DREL, U2DREL, and U2DINT), change the statement

```
      CHARACTER*20 FMTIN
```

   to

```
      DIMENSION FMTIN(5)
```

3. In each of the array reading utility modules (U1DREL, U2DREL, and U2DINT), change all occurrences of "A20" to "5A4". For example, change

```
     1        ' USING FORMAT: ',A20/30X,79('-')/)
```

   in subroutine U1DREL to

```
     1        ' USING FORMAT: ',5A4/30X,79('-')/)
```

4. To make the program strictly comply with the FORTRAN 66 standard, it is necessary to change character constants in DATA statements to Hollerith constants. However, most FORTRAN 66 compilers accept character constants specified using FORTRAN 77 notation (using apostrophes). Thus, it is unlikely that this change will be required.