BRNANL, A Fortran Program to Identify Basic Blocks in Fortran Programs*

by

Lloyd D. Fosdick
Department of Computer Science
University of Colorado
Boulder, Colorado

Report #CU-CS-040-74 March, 1974

* This work supported in part by National Science Foundation Grant GJ-660 and National Science Foundation Grant GJ-36461

Abstract

A basic block is a sequence of consecutive Fortran statements which must be executed consecutively; that is, if one statement in the block is executed, all are executed. Except for special cases noted in the text, a Fortran program is a catenation of basic blocks. BRNANL is a Fortran program designed to recognize basic blocks in a Fortran program. Given a Fortran program (FP) BRNANL will generate a modified Fortran program (MFP) in which a subroutine call is located at the head of every basic block. Execution of the MFP produces the same results as execution of the FP but the inserted subroutine calls permit monitoring of the execution sequences. User information for running BRNANL is presented.

Keywords: Software testing, control path analysis.

1. Introduction

This report describes external features of a program BRNANL which is designed to identify basic blocks in an ANSI Fortran [1] program. Informally, a basic block (BB) is a sequence of statements which must be executed consecutively: a precise definition will be given later. The following example serves to illustrate the idea of a BB in Fortran:

```
          .          }
          .          }
          .          }
       K = K + 1     }
       IF(K) 10, 20, 30 }   tail end of BB

    10 X = X + Y     }      BB

    20 Y = 3.0*Z + 9.0 }
       D = A*B + C*D   }    BB
       GO TO(40, 50), J }

    40 A = 5.0       }      head end of BB
          .          }
          .          }
          .          }
```

The notion of a basic block which we use is similar, but not identical, to that used in the specification of ANSI Fortran ([1], section 10.2.7). It follows more closely the definition generally used in the code optimization literature [2, 3, 4, 5]. Programs similar to this have appeared before. The program which is most similar to ours is one called FETE [6] written by Ingalls*, another program of this type appears to have been contained in a larger program reported by Allen [4]. A slightly different, but related program has been reported by Russell and Estrin [7]. We desired a program of this type, written in ANSI Fortran for portability, which could be easily

* An improved version, called FORTUNE, is commercially available from Capex (Phoenix, Arizona).

modified to meet various needs we had in connection with a project on software validation. For these reasons we created the program described here.

BRNANL accepts as input a syntactically correct ANSI Fortran program, say FP, and produces a modified form of FP, say MFP, differing from FP in that a subroutine call has been placed at the beginning of every BB. BRNANL numbers the BBs in order of their appearance in the source code and this number appears as a parameter in the inserted subroutine call. There is a second calling parameter which is used to identify special situations. A subroutine call is inserted before the first executable statement of a program; the second parameter has the value 1 in this case. A subroutine call is inserted before every STOP statement of a program; the second parameter has the value 3 in this case. In the normal case, when the inserted call appears as the first statement of a BB the second parameter has the value 2. The MFP for the example above is shown below.

```
         .
         .
         .
    K = K + 1
    IF(K) 10, 20, 30
10  CALL XXXXXX(5, 2)
    X = X + Y
20  CALL XXXXXX(6, 2)
    Y = 3.0*Z + 9.0
    D = A*B + C*D
    GO TO (40, 50), J
40  CALL XXXXXX(7, 2)
    A = 5.0
         .
         .
         .
```

Here the first BB of the segment has been arbitrarily numbered 5 and the called subroutine arbitrarily named XXXXXX.

The MFP produced by BRNANL has the physical form of a printed listing and/or a punched deck and/or a file which may be on disc or tape depending on the system under which it executes. All statements inserted by BRNANL in the MFP are flagged by asterisks in columns 73-80 of the output. The printed listing and the file have the block number of each statement and the sequential line number recorded in columns to the right of each statement. A copy of the FP and the MFP are shown in Appendix E for a subroutine subprogram.

It is possible to supress entirely the insertion of the subroutine calls. This is controlled by a datum on a data card read by BRNANL (cf. Appendix A). When this option is used, the listing which is produced will have the block number and line number at the right of each statement as before. This option is used to analyze the structure of the flowgraph for the program. In this report we are primarily concerned with using BRNANL to obtain a MFP which does have the subroutine calls inserted, so no further consideration is given to this option.

When the MFP is executed, various types of information can be recorded depending on the subroutine XXXXXX*. For example, the set of basic blocks executed can be recorded, the frequency of execution of basic blocks can be recorded, the sequence in which basic blocks are executed can be recorded, etc. Used in this way BRNANL is a valuable tool in program testing and it

* Henceforth we will use XXXXXX for the name of the subroutine appearing in the inserted call.

was with this purpose in mind that BRNANL was constructed. Since its construction, we have found it to be a useful tool in the reduction of a Fortran program to a directed graph.

Although BRNANL was written in ANSI Fortran it does contain one machine dependent subroutine CHRCHK which is designed to classify a character which has been read with an A1 format specification as a letter, a digit, or a special character. Specifications of this subroutine will be found in Appendix D.

Use of BRNANL is very simple. One card containing parameter specifications is placed in front of the FP and one card containing the character $ in column 7 is placed in back of the FP: the resulting deck is the data deck for BRNANL. This deck setup is shown in Appendix B. The first card contains the following parameter specifications: name of the subroutine for the inserted call; a unique Fortran variable name (i.e. a name not used in the FP); initial block number; initial line number; flag to indicate suppression of inserted subroutine calls. Details are in Appendix A. The total storage required for the assembled program on the CDC 6400 computer operating under KRONOS 2.1 using the RUN compiler is 4240_8 words. In this environment sixty-six seconds of central processor time was required to run BRNANL on itself which consists of 1737 source statements excluding comments.

2. Basic Blocks

In this section the precise rules for identifying basic blocks and inserting subroutine calls are given. As already indicated, a BB is a sequence of one or more consecutive, executable statements in a source program. Suppose we identify the consecutive executable statements in a source program as $S_1, S_2, \ldots, S_n$. Each BB consists of some subsequence, say $S_j, S_{j+1}, \ldots, S_{j+k}$; the first statement, S_j is called the head, the last statement, S_{j+k}, is called the tail, and the sequence of statements between the head and the tail, $S_{j+1}, S_{j+2}, \ldots, S_{j+k-1}$ is called the trunk. The trunk may be empty and the head and tail may be embodied in a single statement.

A tail is any one of the following:

(a) logical IF statement;

(b) arithmetic IF statement;

(c) DO statement;

(d) any form of GO TO statement;

(e) RETURN statement

(f) STOP statement;

(g) any statement followed by a labelled statement except when the labelled statement is a FORMAT statement or when the labelled statement is the terminal statement in a DO loop;

(h) the terminal statement in a DO loop.

A head is the statement immediately following a tail with two exceptions: the terminal statement in a DO-loop is never a head; the first executable statement of the main program and every subprogram is a head.

From this definition, excepting three special situations discussed below, the following assertions are true:

(a) Every S_i belongs to a BB and cannot belong to more than one BB.

(b) If any S_i in a BB is executed then every statement in that BB is executed.

From the first assertion it follows that we may view a program as a catenation of basic blocks. From the second assertion we may conclude that if the head statement of every BB of the program is executed then every statement of the program is executed.

3. Exceptional Situations

The logical IF statement presents one exceptional situation. Consider the statement:

```
        IF(K.LT.0)  X = X + Y.
```

This statement is a tail, however execution of this tail does not necessarily imply execution of the embedded assignment statement X = X + Y. To resolve this we treat only the structure

```
        IF(<logical expression>)
```

as a tail and the portion of the logical IF following this is treated as a BB consisting of one statement. Thus in the above example IF(K.LT.0) would be the tail for, say, BB(12), then the statement X = X + Y would be BB(13). With this understanding the two assertions above remain valid.

The second exceptional situation arises when a jump within a DO-loop can go to the last statement in the scope of a DO, as illustrated in the following situation

```
            DO 20 J = 1, K
            X(J) = X(J) + Y
            IF(X(J))  10, 20, 30
        10  L = L + 1
        20  V(J) = 0
        30  A = B + C
```

Since there is a jump possible to statement

```
        20  V(J) = 0
```

we ought to identify it as a head -- it is a tail by virtue of it being the terminal statement in a DO-loop (rule (k) above). However, if we were to treat this statement also as a head, then we would have in the MFP

```
        20  CALL XXXXXX(--,--)
            V(J) = 0
```

violating the DO-loop. We resolve this problem by not permitting the terminal statement in a DO-loop to be a head; it is always treated only as a tail. The MFP for the program segment above is

```
      DO 20 J = 1, K
      CALL XXXXXX(12, 2)
      X(J) = X(J) + Y
      IF(X(J))  10, 20, 30
10    CALL XXXXXX(13, 2)
      L = L + 1
20    V(J) = 0
      CALL XXXXXX(14, 2)
30    CALL XXXXXX(15, 2)
      A = B + c
```

where the BB numbering arbitrarily starts at 12 and BB(14) is a dummy used to detect satisfying the DO-loop. Thus the pair of statements

```
10    L = L + 1
20    V(J) = 0
```

is BB(13) and it is evident that assertion (b) above is not true. On the other hand it is important to note that if the head of every BB (including the dummy) is executed, then it is true that every statement has been executed. Also the number of times the statement

```
20    V(J) = 0
```

is executed is given by the expression

$$n_{12} - n_{11} + n_{14}$$

where n_i is the number of times BB(i) is entered. Finally, we observe that if DO-loops are terminated with CONTINUE statements, a good programming practice in any case, then jumps to the end of a DO-loop do not cause any important difficulty since one is not usually interested in the execution of CONTINUE statements.

The third exceptional sitaution arises when there is no return to the calling program after a subprogram has been called into execution. For example, in the basic block,

```
          --
          --
          --
   30    J = J + 1
         X = X + Y
         CALL XAMPL(X, J, Z)
         X = Z
         GO TO 20
          --
          --
          --
```

faiture to return from XAMPL makes assertion (b) above false. If STOP statements are permitted only in the main program, then this situation cannot arise unless execution is aborted by the system due to an error (overflow, array bounds violation, etc.).

These special situations could be eliminated. The logical IF problem could be removed by a replacement of the logical IF by an arithmetic IF and suitable restructuring of the program. The DO-loop problem could be removed by permitting assignment of new statement labels and appropriate re-labelling. Finally, a change in the rules defining a BB could partially eliminate the STOP statement problem. A subroutine CALL statement could be a BB but there would still be a problem with FUNCTION calls since these are embedded in statements.

The splitting of a logical IF statement in two BBs is done in the following way. Suppose we have the logical IF statement

```
<label>  IF(<Boolean expression>)<statement>
```

then in the MFP this appears as

```
<label>  LLLLLL = <Boolean expression>
         IF(LLLLLL) CALL XXXXXX(--,--)
         IF(LLLLLL)<statement>
```

It is evident that the subroutine call will be executed if and only if <statement> is executed so the call can be associated with the BB for the statement. The name LLLLLL is arbitrary: it is the second parameter on the data card. The MFP will also have a type declaration:

```
LOGICAL  LLLLLL
```

Since a logical IF can terminate a DO-loop, it is evident that this situation presents a special problem. Splitting of the logical IF is not done in this case. Thus in the following sequence

```
        DO 10 I = 1, N
        --
        --
        --
   10   IF(X.LT.0) X = 1
```

The "BB" X = 1 is not identified as a BB. If the assertion (b) is to be valid, it is evident that a logical IF terminating a DO-loop must be prohibited. When BRNANL detects this situation it prints an error message.

In our own use of BRNANL we preprocess the FP with another program, STYLE [8], which reformats the FP and causes each DO loop to terminate on a CONTINUE statement. This essentially removes the difficulties cited above.

4. First Executable Statement and STOP Statement.

The first BB executed in a program is given special treatment. Suppose the FP begins with the statements

```
C THIS IS THE MAIN PROGRAM
      DIMENSION A(10), B(10, 10)
 10   READ (5, 999) A
      DO 20 I = 1, 10
      --
      --
      --
```

Then the MFP begins with the statements

```
C THIS IS THE MAIN PROGRAM
      DIMENSION A(10),B(10, 10)
      CALL XXXXXX(0, 1)
 10   CALL XXXXXX(1, 2)
      READ(5, 999) A
      DO 20 I = 1, 10
```

The second parameter of the first CALL is 1, uniquely identifying it as preceding the first executable statement in the program. This information allows the routine XXXXXX to perform initialization. The first parameter in this call is one less than the initial block number, the third parameter on the data card (cf. Appendix A); here it is assumed that this number was 1. The second CALL, which would be there even if the label were not on the READ statement, is used to identify actual entry into the BB. It is to be noted that this situation arises only in a main program.

The BB in which a STOP appears as the tail is treated in a special way. In addition to the call which is inserted at the head of the block, a call is inserted immediately before the STOP statement. The following example illustrates this. Suppose the FP contains the BB, say BB(100),

```
150   X = SIN(Y)
      WRITE(6, 999) X, Y
      STOP
```

Then the MFP is

```
150   CALL XXXXXX(100, 2)
      X = SIN(Y)
      WRITE(6, 999) X, Y
      CALL XXXXXX(100, 3)
      STOP
```

It is to be noted that the second parameter in the CALL just before the STOP is 3; this special value is used only before a STOP so the routine XXXXXX can take whatever steps are appropriate for such a condition. Typically it would print accumulated data on BB activity in executing the MFP.

5. Limitations.

The most important limitation arises from the fact that BRNANL assumes that the FP is a syntactically correct ANSI Fortran program; if it is not, incorrect execution may result.

Other limitations are listed below:

1. Maximum number that can be assigned to a BB is 9999;
2. Maximum number of subscripted variables in each program unit (subroutine subprogram, function subprogram, main program) is 50;
3. Maximum depth for DO-loop nesting is 29.

It is recommended that all DO-loops terminate on CONTINUE statements. (Preprocessing the FP by STYLE [8] will guarantee this.) If a DO-loop does not terminate on a CONTINUE statement BRNANL will still execute properly, however jumps to the last statement in the DO need special consideration as described in section 3 of this report.

A list of error messages which can be produced by BRNANL is given in Appendix B.

6. I/O FILES.

BRNANL reads the input file from unit 5, writes the print file on unit 6, and writes the punch file on unit 7. Specifically all READ statements have the form

```
READ(KIN, ...
```

all WRITE statements for producing the listing of the MFP and any error messages have the form

```
WRITE(KPR, ...
```

and all write statements for producing the source "deck" for the MFP have the form

```
WRITE(KPU, ...
```

A DATA statement is used assign 5, 6, 7 to KIN, KPR, KPU, respectively.

7. Acknowledgements.

Much of the testing, corrections to errors in the original program, and insertion of COMMENTS was done by Jeffery Wright and Jacob Wu, graduate students in the Department of Computer Science at the University of Colorado.

8. References.

1. American National Standard Fortran, American National Standards Inc. (1966).
   See also:
   (a) Fortran vs. Basic Fortran, Comm ACM 7 (Oct. 1964), 591-625.
   (b) Clarification of Fortran Standards - Initial Progress, Comm A M 12 (May 1969), 289-294.
   (c) Clarification of Fortran Standards - Second Report, Comm ACM 14 (Oct 1971), 628-642.

2. J. Cocke and J. T. Schwartz, Programming Languages and Their Compilers, Courant Institute, NYU (1970).

3. F. E. Allen, Program optimization, in Annual Review in Automatic Programming, Vol. 5, Pergamon (1969).

4. F. E. Allen, Control flow analysis. ACM SIGPLAN Notices 5, 7 (1970), 1-19.

5. A. Aho and J. D. Ullman, The Theory of Parsing, Translation and Compiling, Volume II: Compiling: Prentice-Hall (1973).

6. D. H. H. Ingalls, FETE, A Fortran execution time estimator, Department of Computer Science, Stanford Univeristy, Report 71-204 (1971).

7. E. C. Russell and G. Estrin, Measurement based automatic analysis of Fortran programs. SJCC (1969), 723-732.

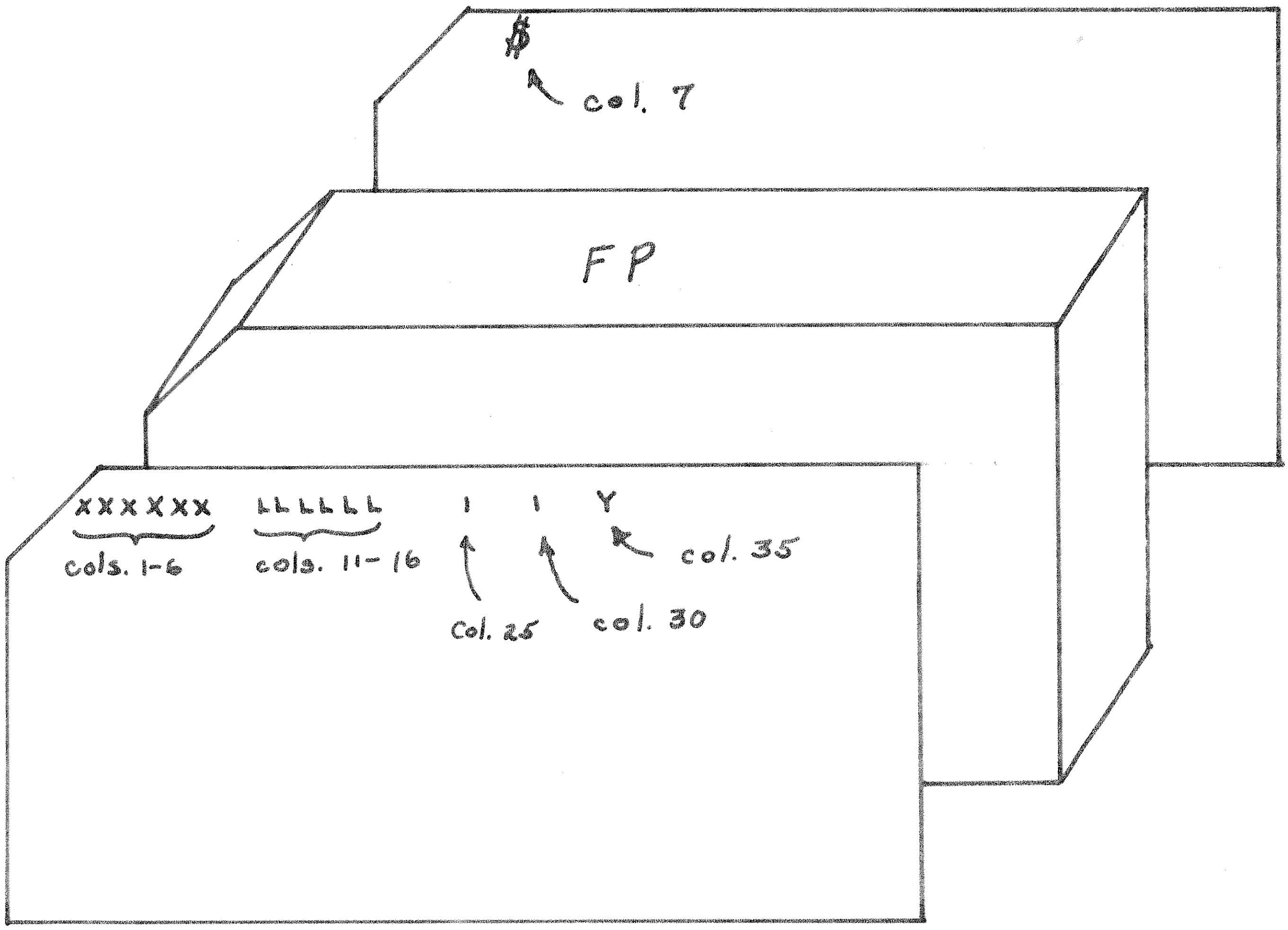
8. Dorothy Lang Wedel, STYLE Editor: User's Guide, Department of Computer Science, University of Colorado, Report 7 (1972).

Appendix A: Data card.

One data card must precede the FP. The layout of this card follows:

cols.	contents
1-6	Name of subroutine appearing in the inserted CALL statements;
11-16	Name of variable used to hold value of Boolean expressions appearing in logical IF statements;
22-25	Number of first BB in program; BBs are numbered sequentially in order of appearance in the FP; value entered as a right-justified integer;
26-30	Number of first line for sequential numbering of lines in output file supplied by BRNANL; value entered as a right-justified integer.
35	Y if subroutine calls are to be inserted.

Appendix B: Run Deck Organization



Appendix C: Error Messages

1: MORE THAN 20 CARDS USED FOR STATEMENT.

The maximum number of cards permitted for a statement is 20 (i.e. 19 continuation cards). Fatal error.

2: IF STATEMENT SYNTAX ERROR.

ANSI Fortran syntax error. Fatal error.

3: MORE THAN 50 DIMENSIONED VARIABLES.

An array in BRNANL called ARRNAM holds the list of dimensioned variables in the subprogram, or main program being processed. It is dimensioned at 50. Fatal error.

4: STATEMENT ENDS WITH LEFT PARENTHESIS.

ANSI Fortran syntax error. Fatal error.

5: DECLARATION FOLLOWED ONLY BY BLANKS.

ANSI Fortran syntax error. Fatal error.

6: LIMIT OF 9999 ON BASIC BLOCK INDEX EXCEEDED.

BRNANL requires BB index to lie in the interval (1, 9999). Fatal error.

7: NON-ANSI BLANK CARD ENCOUNTERED.

Blank cards are not permitted in the subject program. This card is automatically replaced by a blank comment card. Non-fatal error.

8: NON-ANSI PROGRAM CARD ENCOUNTERED.

A PROGRAM card is required for the main program in CDC Fortran, however this is not legal ANSI Fortran. This card is ignored. Non-fatal error.

9: LABEL CONTAINS AN ILLEGAL CHARACTER.

A label must consist of digits only. Fatal error.

10: LOGICAL IF CLOSING A DO-LOOP.

After each error message the following information is printed:

```
BUFFER A CONTAINS

   ---(statement being processed)

BUFFER B CONTAINS

   ---(next statement to be processed)

BUFFER D CONTAINS

   ---(first card of next statement)
```

Appendix D: Machine Dependent Subroutine

The subroutine CHRCHK in BRNANL is machine dependent and may have to be modified by the user for systems other than the CDC 6400. This subroutine determines whether a character is a letter, a digit, or special.

The subroutine specification is

```
SUBROUTINE CHRCHK(A, I, L)
```

where the formal parameters are defined as follows:

A -- a one dimensional array holding characters which have been read into it using an A1 format specification.

I -- the character to be checked is in position A(I).

L -- the routine CHRCHK makes the assignment

L = 1 if the character in A(I) is a letter

L = 2 if the character in A(I) is a digit

L = 3 if the character in A(I) is special.

A listing of this subroutine is on the following page.

```
      SUBROUTINE CHRCHK(A, I, L)
C THIS IS A CHARACTER CHECK ROUTINE FOR BUFFER A. L=1 IF
C A(I) IS A LETTER, L=2 IF A(I) IS A DIGIT,L=3 IF A(I) IS A
C SPECIAL CHARACTER.
      DIMENSION A(1)
      INTEGER ALPBL, ALPBH, NUML, NUMH, A
C  THE FOLLOWING CHECKING TECHNIQUE SHOULD BE ADJUSTED FOR
C LOCAL CHARACTER SET IF ALPHABET CHARACTERS OR NUMERIC
C CHARACTERS ARE NOT
C   IN A CONTIGUOUS GROUP.
      DATA ALPBL /1HA/, ALPBH /1HZ/, NUML /1H0/, NUMH /1H9/
      ICHR = A(I)
      IF (ICHR-ALPBL) 20, 10, 10
   10 IF (ICHR-ALPBH) 40, 40, 20
   20 IF (ICHR-NUML) 60, 30, 30
   30 IF (ICHR-NUMH) 50, 50, 60
   40 L = 1
      RETURN
   50 L = 2
      RETURN
   60 L = 3
      RETURN
      END
```

Appendix E: Example

On the following six pages an example illustrating the output obtained from BRNANL is shown. The first three pages contain the listing of the FP (a subroutine subprogram KZEONE). The next three pages contain the MFP as contained on the print file. In this example block numbering starts at 1 and line numbering start, at 10. Block numbers associated with logical IF statements are flagged by an asterisk.

```
      SUBROUTINE KZEONE(X, Y, RE0, IM0, RE1, IM1)
C THE VARIABLES X AND Y ARE THE REAL AND IMAGINARY PARTS OF
C THE ARGUMENT OF THE FIRST TWO MODIFIED BESSEL FUNCTIONS
C OF THE SECOND KIND,K0 AND K1.  RE0,IM0,RE1 AND IM1 GIVE
C THE REAL AND IMAGINARY PARTS OF EXP(X)*K0 AND EXP(X)*K1,
C RESPECTIVELY.  ALTHOUGH THE REAL NOTATION USED IN THIS
C SUBROUTINE MAY SEEM INELEGANT WHEN COMPARED WITH THE
C COMPLEX NOTATION THAT FORTRAN ALLOWS, THIS VERSION RUNS
C ABOUT 30 PERCENT FASTER THAN ONE WRITTEN USING COMPLEX
C VARIABLES.
      DOUBLE PRECISION X, Y, X2, Y2, RE0, IM0, RE1, IM1,
     * R1, R2, T1, T2, P1, P2, RTERM, ITERM, EXSQ(8), TSQ(8)
      DATA TSQ(1) /0.0D0/, TSQ(2) /3.19303633920635D-1/,
     * TSQ(3) /1.29075862295915D0/, TSQ(4)
     * /2.95837445869665D0/, TSQ(5) /5.40903159724444D0/,
     * TSQ(6) /8.80407957805676D0/, TSQ(7)
     * /1.34685357432515D1/, TSQ(8) /2.02499163658709D1/,
     * EXSQ(1) /0.564100308726400/, EXSQ(2)
     * /0.412028687498900/, EXSQ(3) /0.158488915795900/,
     * EXSQ(4) /0.307800338725500-1/, EXSQ(5)
     * /0.277806884291300-2/, EXSQ(6) /0.100004441232500-3/,
     * EXSQ(7) /0.105911554771100-5/, EXSQ(8)
     * /0.152247580425400-8/
C THE ARRAYS TSQ AND EXSQ CONTAIN THE SQUARE OF THE
C ABSCISSAS AND THE WEIGHT FACTORS USED IN THE GAUSS-
C HERMITE QUADRATURE.
      R2 = X*X + Y*Y
      IF (X.GT.0.0D0 .OR. R2.NE.0.0D0) GO TO 10
      WRITE (6,99999)
      RETURN
   10 IF (R2.GE.1.96D2) GO TO 50
      IF (R2.GE.1.849D1) GO TO 30
C THIS SECTION CALCULATES THE FUNCTIONS USING THE SERIES
C EXPANSIONS
      X2 = X/2.0D0
      Y2 = Y/2.0D0
      P1 = X2*X2
      P2 = Y2*Y2
      T1 = -(DLOG(P1+P2)/2.0D0+0.5772156649015329D0)
C THE CONSTANT IN THE PRECEDING STATEMENT IS EULER*S
C CONSTANT
      T2 = -DATAN2(Y,X)
      X2 = P1 - P2
      Y2 = X*Y2
      RTERM = 1.0D0
      ITERM = 0.0D0
      RE0 = T1
      IM0 = T2
      T1 = T1 + 0.5D0
      RE1 = T1
      IM1 = T2
      P2 = DSQRT(R2)
      L = 2.106D0*P2 + 4.4D0
      IF (P2.LT.8.0D-1) L = 2.129D0*P2 + 4.0D0
      DO 20 N=1,L
        P1 = N
        P2 = N*N
        R1 = RTERM
        RTERM = (R1*X2-ITERM*Y2)/P2
        ITERM = (R1*Y2+ITERM*X2)/P2
        T1 = T1 + 0.5D0/P1
        RE0 = RE0 + T1*RTERM - T2*ITERM
        IM0 = IM0 + T1*ITERM + T2*RTERM
      P1 = P1 + 1.0D0
      T1 = T1 + 0.5D0/P1
      RE1 = RE1 + (T1*RTERM-T2*ITERM)/P1
      IM1 = IM1 + (T1*ITERM+T2*RTERM)/P1
   20 CONTINUE
      R1 = X/R2 - 0.5D0*(X*RE1-Y*IM1)
      R2 = -Y/R2 - 0.5D0*(X*IM1+Y*RE1)
      P1 = DEXP(X)
      RE0 = P1*RE0
      IM0 = P1*IM0
      RE1 = P1*R1
      IM1 = P1*R2
      RETURN
C THIS SECTION CALCULATES THE FUNCTIONS USING THE INTEGRAL
C REPRESENTATION, EQN 3, EVALUATED WITH 15 POINT GAUSS-
C HERMITE QUADRATURE
   30 X2 = 2.0D0*X
      Y2 = 2.0D0*Y
      R1 = Y2*Y2
      P1 = DSQRT(X2*X2+R1)
      P2 = DSQRT(P1+X2)
      T1 = EXSQ(1)/(2.0D0*P1)
      RE0 = T1*P2
      IM0 = T1/P2
      RE1 = 0.0D0
      IM1 = 0.0D0
      DO 40 N=2,8
         T2 = X2 + TSQ(N)
         P1 = DSQRT(T2*T2+R1)
         P2 = DSQRT(P1+T2)
         T1 = EXSQ(N)/P1
         RE0 = RE0 + T1*P2
         IM0 = IM0 + T1/P2
         T1 = EXSQ(N)*TSQ(N)
         RE1 = RE1 + T1*P2
         IM1 = IM1 + T1/P2
   40 CONTINUE
      T2 = -Y2*IM0
      RE1 = RE1/R2
      R2 = Y2*IM1/R2
      RTERM = 1.41421356237309D0*DCOS(Y)
      ITERM = -1.41421356237309D0*DSIN(Y)
C THE CONSTANT IN THE PREVIOUS STATEMENTS IS,OF COURSE,
C SQRT(2.0).
      IM0 = RE0*ITERM + T2*RTERM
      RE0 = RE0*RTERM - T2*ITERM
      T1 = RE1*RTERM - R2*ITERM
      T2 = RE1*ITERM + R2*RTERM
      RE1 = T1*X + T2*Y
      IM1 = -T1*Y + T2*X
      RETURN
C THIS SECTION CALCULATES THE FUNCTIONS USING THE
C ASYMPTOTIC EXPANSIONS
   50 RTERM = 1.0D0
      ITERM = 0.0D0
      RE0 = 1.0D0
      IM0 = 0.0D0
      RE1 = 1.0D0
      IM1 = 0.0D0
      P1 = 8.0D0*R2
      P2 = DSQRT(R2)
      L = 3.91D0+8.12D1/P2
      R1 = 1.0D0
      R2 = 1.0D0
      M = -8
      K = 3
      CO 60 N=1,L
         M = M + 8
         K = K - M
         R1 = FLOAT(K-4)*R1
         R2 = FLOAT(K)*R2
         T1 = FLOAT(N)*P1
         T2 = RTERM
         RTERM = (T2*X+ITERM*Y)/T1
         ITERM = (-T2*Y+ITERM*X)/T1
         RE0 = RE0 + R1*RTERM
         IM0 = IM0 + R1*ITERM
         RE1 = RE1 + R2*RTERM
         IM1 = IM1 + R2*ITERM
   60 CONTINUE
      T1 = DSQRT(P2+X)
      T2 = -Y/T1
      P1 = 8.8622692545275801D-1/P2
C THIS CONSTANT IS SQRT(PI)/2.0, WITH PI=3.14159...
      RTERM = P1*DCOS(Y)
      ITERM = -P1*DSIN(Y)
      R1 = RE0*RTERM - IM0*ITERM
      R2 = RE0*ITERM + IM0*RTERM
      RE0 = T1*R1 - T2*R2
      IM0 = T1*R2 + T2*R1
      R1 = RE1*RTERM - IM1*ITERM
      R2 = RE1*ITERM + IM1*RTERM
      RE1 = T1*R1 - T2*R2
      IM1 = T1*R2 + T2*R1
      RETURN
99999 FORMAT (42H  ARGUMENT OF THE BESSEL FUNCTIONS IS ZERO,
     * 35H OR LIES IN LEFT HALF COMPLEX PLANE)
      END
      SUBROUTINE KZEONE(X, Y, RE0, IM0, RE1, IM1)                       KZE  10     0
      LOGICAL LLLLLL                                                    ********    0
C THE VARIABLES X AND Y ARE THE REAL AND IMAGINARY PARTS OF             KZE  20     0
C THE ARGUMENT OF THE FIRST TWO MODIFIED BESSEL FUNCTIONS               KZE  30     0
C OF THE SECOND KIND,K0 AND K1.  RE0,IM0,RE1 AND IM1 GIVE               KZE  40     0
C THE REAL AND IMAGINARY PARTS OF EXP(X)*K0 AND EXP(X)*K1,              KZE  50     0
C RESPECTIVELY.  ALTHOUGH THE REAL NOTATION USED IN THIS                KZE  60     0
C SUBROUTINE MAY SEEM INELEGANT WHEN COMPARED WITH THE                  KZE  70     0
C COMPLEX NOTATION THAT FORTRAN ALLOWS, THIS VERSION RUNS               KZE  80     0
C ABOUT 30 PERCENT FASTER THAN ONE WRITTEN USING COMPLEX                KZE  90     0
C VARIABLES.                                                            KZE 100     0
      DOUBLE PRECISION X, Y, X2, Y2, RE0, IM0, RE1, IM1,                KZE 110     0
     * R1, R2, T1, T2, P1, P2, RTERM, ITERM, EXSQ(8), TSQ(8)            KZE 120     0
      DATA TSQ(1) /0.0D0/, TSQ(2) /3.19303633920635D-1/,                KZE 130     0
     * TSQ(3) /1.29075862295915D0/, TSQ(4)                              KZE 140     0
     * /2.95837445869665D0/, TSQ(5) /5.40903159724444D0/,               KZE 150     0
     * TSQ(6) /8.80407957805676D0/, TSQ(7)                              KZE 160     0
     * /1.34685357432515D1/, TSQ(8) /2.02499163658709D1/,               KZE 170     0
     * EXSQ(1) /0.5641003087264D0/, EXSQ(2)                             KZE 180     0
     * /0.4120286874989D0/, EXSQ(3) /0.1584889157959D0/,                KZE 190     0
     * EXSQ(4) /0.3078003387255D-1/, EXSQ(5)                            KZE 200     0
     * /0.2778068842913D-2/, EXSQ(6) /0.1000044412325D-3/,              KZE 210     0
     * EXSQ(7) /0.1059115547711D-5/, EXSQ(8)                            KZE 220     0
     * /0.1522475804254D-8/                                             KZE 230     0
C THE ARRAYS TSQ AND EXSQ CONTAIN THE SQUARE OF THE                      KZE 240     0
C ABSCISSAS AND THE WEIGHT FACTORS USED IN THE GAUSS-                   KZE 250     0
C HERMITE QUADRATURE.                                                   KZE 260     0
      CALL XXXXXX(0001,2)                                               ********    1
      R2 = X*X + Y*Y                                                    KZE 270     1
      LLLLLL=X.GT.0.0D0.OR.R2.NE.0.0D0                                  ********    1
      IF(LLLLLL)CALL XXXXXX(0002,2)                                     ********    2*
      IF(LLLLLL)                         GO TO 10                       KZE 280     2*
      CALL XXXXXX(0003,2)                                               ********    3
      WRITE (6,99999)                                                   KZE 290     3
      RETURN                                                            KZE 300     3
   10 CALL XXXXXX(0004,2)                                               ********    4
      LLLLLL=R2.GE.1.96D2                                               ********    4
      IF(LLLLLL)CALL XXXXXX(0005,2)                                     ********    5*
      IF(LLLLLL)          GO TO 50                                      KZE 310     5*
      CALL XXXXXX(0006,2)                                               ********    6
      LLLLLL=R2.GE.1.849D1                                              ********    6
      IF(LLLLLL)CALL XXXXXX(0007,2)                                     ********    7*
      IF(LLLLLL)          GO TO 30                                      KZE 320     7*
C THIS SECTION CALCULATES THE FUNCTIONS USING THE SERIES                KZE 330     7
C EXPANSIONS                                                            KZE 340     7
      CALL XXXXXX(0008,2)                                               ********    8
      X2 = X/2.0D0                                                      KZE 350     8
      Y2 = Y/2.0D0                                                      KZE 360     8
      P1 = X2*X2                                                        KZE 370     8
      P2 = Y2*Y2                                                        KZE 380     8
      T1 = -(DLOG(P1+P2)/2.0D0+0.5772156649015329D0)                    KZE 390     8
C THE CONSTANT IN THE PRECEDING STATEMENT IS EULER*S                    KZE 400     8
C CONSTANT                                                              KZE 410     8
      T2 = -DATAN2(Y,X)                                                 KZE 420     8
      X2 = P1 - P2                                                      KZE 430     8
      Y2 = X*Y2                                                         KZE 440     8
      RTERM = 1.0D0                                                     KZE 450     8
      ITERM = 0.0D0                                                     KZE 460     8
      RE0 = T1                                                          KZE 470     8
      IM0 = T2                                                          KZE 480     8
      T1 = T1 + 0.5D0                                                   KZE 490     8
      RE1 = T1                                                          KZE 500      8
      IM1 = T2                                                          KZE 510      8
      P2 = DSQRT(R2)                                                    KZE 520      8
      L = 2.106D0*P2 + 4.4D0                                            KZE 530      8
      LLLLLL=P2.LT.8.0D-1                                               ********     8
      IF(LLLLLL)CALL XXXXXX(0009,2)                                     ********     9*
      IF(LLLLLL)        L = 2.129D0*P2 + 4.0D0                          KZE 540      9*
      CALL XXXXXX(0010,2)                                               ********    10
      DO 20 N=1,L                                                       KZE 550     10
      CALL XXXXXX(0011,2)                                               ********    11
         P1 = N                                                         KZE 560     11
         P2 = N*N                                                       KZE 570     11
         R1 = RTERM                                                     KZE 580     11
         RTERM = (R1*X2-ITERM*Y2)/P2                                    KZE 590     11
         ITERM = (R1*Y2+ITERM*X2)/P2                                    KZE 600     11
         T1 = T1 + 0.5D0/P1                                             KZE 610     11
         RE0 = RE0 + T1*RTERM - T2*ITERM                                KZE 620     11
         IM0 = IM0 + T1*ITERM + T2*RTERM                                KZE 630     11
         P1 = P1 + 1.0D0                                                KZE 640     11
         T1 = T1 + 0.5D0/P1                                             KZE 650     11
         RE1 = RE1 + (T1*RTERM-T2*ITERM)/P1                             KZE 660     11
         IM1 = IM1 + (T1*ITERM+T2*RTERM)/P1                             KZE 670     11
   20 CONTINUE                                                          KZE 680     11
      CALL XXXXXX(0012,2)                                               ********    12
      R1 = X/R2 - 0.5D0*(X*RE1-Y*IM1)                                   KZE 690     12
      R2 = -Y/R2 - 0.5D0*(X*IM1+Y*RE1)                                  KZE 700     12
      P1 = DEXP(X)                                                      KZE 710     12
      RE0 = P1*RE0                                                      KZE 720     12
      IM0 = P1*IM0                                                      KZE 730     12
      RE1 = P1*R1                                                       KZE 740     12
      IM1 = P1*R2                                                       KZE 750     12
      RETURN                                                            KZE 760     12
C THIS SECTION CALCULATES THE FUNCTIONS USING THE INTEGRAL               KZE 770     12
C REPRESENTATION, EQN 3, EVALUATED WITH 15 POINT GAUSS-                  KZE 780     12
C HERMITE QUADRATURE                                                     KZE 790     12
   30 CALL XXXXXX(0013,2)                                               ********    13
      X2 = 2.0D0*X                                                      KZE 800     13
      Y2 = 2.0D0*Y                                                      KZE 810     13
      R1 = Y2*Y2                                                        KZE 820     13
      P1 = DSQRT(X2*X2+R1)                                              KZE 830     13
      P2 = DSQRT(P1+X2)                                                 KZE 840     13
      T1 = EXSQ(1)/(2.0D0*P1)                                           KZE 850     13
      RE0 = T1*P2                                                       KZE 860     13
      IM0 = T1/P2                                                       KZE 870     13
      RE1 = 0.0D0                                                       KZE 880     13
      IM1 = 0.0D0                                                       KZE 890     13
      DO 40 N=2,8                                                       KZE 900     13
      CALL XXXXXX(0014,2)                                               ********    14
         T2 = X2 + TSQ(N)                                               KZE 910     14
         P1 = DSQRT(T2*T2+R1)                                           KZE 920     14
         P2 = DSQRT(P1+T2)                                              KZE 930     14
         T1 = EXSQ(N)/P1                                                KZE 940     14
         RE0 = RE0 + T1*P2                                              KZE 950     14
         IM0 = IM0 + T1/P2                                              KZE 960     14
         T1 = EXSQ(N)*TSQ(N)                                            KZE 970     14
         RE1 = RE1 + T1*P2                                              KZE 980     14
         IM1 = IM1 + T1/P2                                              KZE 990     14
   40 CONTINUE                                                          KZE1000     14
      CALL XXXXXX(0015,2)                                               ********    15
      T2 = -Y2*IM0                                                      KZE1010     15
      RE1 = RE1/R2                                                      KZE1020     15
      R2 = Y2*IM1/R2                                                    KZE1030     15
      RTERM = 1.41421356237309D0*DCOS(Y)                                KZE1040     15
      ITERM = -1.4142135623730900*DSIN(Y)                              KZE1050       15
C THE CONSTANT IN THE PREVIOUS STATEMENTS IS.OF COURSE.                KZE1060       15
C SQRT(2.0).                                                           KZE1070       15
      IM0 = RE0*ITERM + T2*RTERM                                       KZE1080       15
      RE0 = RE0*RTERM - T2*ITERM                                       KZE1090       15
      T1 = RE1*RTERM - R2*ITERM                                        KZE1100       15
      T2 = RE1*ITERM + R2*RTERM                                        KZE1110       15
      RE1 = T1*X + T2*Y                                                KZE1120       15
      IM1 = -T1*Y + T2*X                                               KZE1130       15
      RETURN                                                           KZE1140       15
C THIS SECTION CALCULATES THE FUNCTIONS USING THE                      KZE1150       15
C ASYMPTOTIC EXPANSIONS                                                KZE1160       15
   50 CALL XXXXXX(0016,2)                                              ********      16
      RTERM = 1.0D0                                                    KZE1170       16
      ITERM = 0.0D0                                                    KZE1180       16
      RE0 = 1.0D0                                                      KZE1190       16
      IM0 = 0.0D0                                                      KZE1200       16
      RE1 = 1.0D0                                                      KZE1210       16
      IM1 = 0.0D0                                                      KZE1220       16
      P1 = 8.0D0*R2                                                    KZE1230       16
      P2 = DSQRT(R2)                                                   KZE1240       16
      L = 3.91D0+8.12D1/P2                                             KZE1250       16
      R1 = 1.0D0                                                       KZE1260       16
      R2 = 1.0D0                                                       KZE1270       16
      M = -8                                                           KZE1280       16
      K = 3                                                            KZE1290       16
      DO 60 N=1,L                                                      KZE1300       16
      CALL XXXXXX(0017,2)                                              ********      17
         M = M + 8                                                     KZE1310       17
         K = K - M                                                     KZE1320       17
         R1 = FLOAT(K-4)*R1                                            KZE1330       17
         R2 = FLOAT(K)*R2                                              KZE1340       17
         T1 = FLOAT(N)*P1                                              KZE1350       17
         T2 = RTERM                                                    KZE1360       17
         RTERM = (T2*X+ITERM*Y)/T1                                     KZE1370       17
         ITERM = (-T2*Y+ITERM*X)/T1                                    KZE1380       17
         RE0 = RE0 + R1*RTERM                                          KZE1390       17
         IM0 = IM0 + R1*ITERM                                          KZE1400       17
         RE1 = RE1 + R2*RTERM                                          KZE1410       17
         IM1 = IM1 + R2*ITERM                                          KZE1420       17
   60 CONTINUE                                                         KZE1430       17
      CALL XXXXXX(0018,2)                                              ********      18
      T1 = DSQRT(P2+X)                                                 KZE1440       18
      T2 = -Y/T1                                                       KZE1450       18
      P1 = 8.8622692545275801-1/P2                                     KZE1460       18
C THIS CONSTANT IS SQRT(PI)/2.0, WITH PI=3.14159...                    KZE1470       18
      RTERM = P1*DCOS(Y)                                               KZE1480       18
      ITERM = -P1*DSIN(Y)                                              KZE1490       18
      R1 = RE0*RTERM - IM0*ITERM                                       KZE1500       18
      R2 = RE0*ITERM + IM0*RTERM                                       KZE1510       18
      RE0 = T1*R1 - T2*R2                                              KZE1520       18
      IM0 = T1*R2 + T2*R1                                              KZE1530       18
      R1 = RE1*RTERM - IM1*ITERM                                       KZE1540       18
      R2 = RE1*ITERM + IM1*RTERM                                       KZE1550       18
      RE1 = T1*R1 - T2*R2                                              KZE1560       18
      IM1 = T1*R2 + T2*R1                                              KZE1570       18
      RETURN                                                           KZE1580       18
99999 FORMAT (42H  ARGUMENT OF THE BESSEL FUNCTIONS IS ZERO,           KZE1590       18
     * 35H OR LIES IN LEFT HALF COMPLEX PLANE)                         KZE1600       18
      END                                                              KZE1610       18
```