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*X + 9.0  \}  BB
  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*A + 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 suppress 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 CHRKCH 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 42408 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 concatenation 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:

\[ \text{IF(K.LT.0) } X = X + Y. \]

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

\[ \text{IF(<logical expression>)} \]

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 situation 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
```

failure 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 relabelling. 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 BR 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

\texttt{READ(KIN, ...)}

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

\texttt{WRITE(KPR, ...)}

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

\texttt{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.

   See also:
   (a) Fortran vs. Basic Fortran, Comm ACM 7 (Oct. 1964), 591-625.


Appendix A: Data card.

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

<table>
<thead>
<tr>
<th>cols.</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>Name of subroutine appearing in the inserted CALL statements;</td>
</tr>
<tr>
<td>11-16</td>
<td>Name of variable used to hold value of Boolean expressions appearing in logical IF statements;</td>
</tr>
<tr>
<td>22-25</td>
<td>Number of first BB in program; BBs are numbered sequentially in order of appearance in the FP; value entered as a right-justified integer;</td>
</tr>
<tr>
<td>26-30</td>
<td>Number of first line for sequential numbering of lines in output file supplied by BRNANL; value entered as a right-justified integer.</td>
</tr>
<tr>
<td>35</td>
<td>Y if subroutine calls are to be inserted.</td>
</tr>
</tbody>
</table>
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 CHRCNK 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 AI format specification.
- **I** -- the character to be checked is in position A(I).
- **L** -- the routine CHRCNK makes the assignment
  
  \[
  L = 1 \text{ if the character in A(I) is a letter} \\
  L = 2 \text{ if the character in A(I) is a digit} \\
  L = 3 \text{ if the character in A(I) is special.}
  \]

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(I)
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/
ICH  = A(I)
  IF (ICH-ALPBL) 20, 10, 10
  10 IF (ICH-ALPBH) 40, 40, 20
  20 IF (ICH-NUML) 60, 30, 30
  30 IF (ICH-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 KZGONE(X, Y, R10, IMO, REL, IM1)
C THE VARIABLES X AND Y ARE THE REAL AND IMAGINARY PARTS OF
C THE ARGUMENT OF THE FIRST TWO MODIFIED HESSL FUNCTIONS
C OF THE SECOND KIND X0 AND K1. R10, IM0, REL 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 FORTHAN ALLOW THIS VERSION RUNS
C ABOUT 30 PERCENT FASTER THAN ONE WRITTEN USING COMPLEX
C VARIABLES.
C
DOUBLE PRECISION X, Y, X2, Y2, R10, IM0, REL, IM1,
* K0, K1, P2, P1, P2, ATERM, ITERM, EX50(8), TSQ(8)
DATA TSQ(1) /1.20971878622459150000E+15/, TSQ(2) /3.18303633926062500000E+13/,
* TSQ(3) /3.12905786622951500000E+12/, TSQ(4) /2.95983744586965000000E+11/,
* TSQ(5) /2.90407957056570000000E+10/, TSQ(6) /2.80407957056570000000E+9/,
* TSQ(7) /2.65635735742515000000E+8/, TSQ(8) /2.44996163658709000000E+7/,
* EX50(11) /1.05403155182924200000/, EX50(12) /1.54849289157990900000/,
* EX50(13) /1.53849289157990900000/, EX50(14) /1.30780082399132000000/,
* EX50(15) /1.00000000000000000000/, EX50(16) /7.10506010885634000000/,
* EX50(17) /3.98021685749890000000/, EX50(18) /6.89416159759900000000/,
* EX50(19) /4.67870088541937000000/, EX50(20) /2.37785000000000000000/,
* EX50(21) /1.00000000000000000000/, EX50(22) /1.00000000000000000000/,
* EX50(23) /1.00000000000000000000/, EX50(24) /1.00000000000000000000/,
* /1.00000000000000000000/, EX50(25) /1.00000000000000000000/, EX50(26) /1.00000000000000000000/,
* /1.00000000000000000000/, EX50(27) /1.00000000000000000000/, EX50(28) /1.00000000000000000000/,
* /1.00000000000000000000/, EX50(29) /1.00000000000000000000/, EX50(30) /1.00000000000000000000/,
C THE VARIATIES TSQ AND EX50 CONTAIN THE SQUARE OF THE
C ABSOSSSAS AND THE WEIGHT FACTORS USED IN THE GAUSS-
C LWRRITE QUADRATURE.
C G2 = X*X + Y*Y
IF [X GTE 0.0000 OR Y2 NE 0.0000] GO TO 10
RETURN
C THE SECTION CALCULATES THE FUNCTIONS USING THE SERIES
C EXPANSIONS
C T1 = -UATAN2(Y,X)
X2 = P1 - P2
Y2 = X*Y2
TERM = 1.0000
ITERM = 0.0000
REU = TI1
IMO = T2
TI = T1 + 0.5000
REL = T1
IM1 = T2
P2 = USQR1(P2)
L = 2.106000*P2 + 0.0000
IF (P2 LT +0.0000-1) L = 2.106000*P2 + 0.0000
C020 N=1
P1 = N
P2 = NPN
HI = HTERM
ITERM = (R1*X2 - ITERM*Y2)/P2
TERM = (R1*Y2 - ITERM*X2)/P2
T1 = TI + 0.5000/P1
REO = ROE + TIMATERM + T2*ITERM
IMO = IM0 + TIMITERM + T3*ITERM
C30 KZE 10
KZE 20
KZE 30
KZE 40
KZE 50
KZE 60
KZE 70
KZE 80
KZE 90
KZE 100
KZE 110
KZE 120
KZE 130
KZE 140
KZE 150
KZE 160
KZE 170
KZE 180
KZE 190
KZE 200
KZE 210
KZE 220
KZE 230
KZE 240
KZE 250
KZE 260
KZE 270
KZE 280
KZE 290
KZE 300
KZE 310
KZE 320
KZE 330
KZE 340
KZE 350
KZE 360
KZE 370
KZE 380
KZE 390
KZE 400
KZE 410
KZE 420
KZE 430
KZE 440
KZE 450
KZE 460
KZE 470
KZE 480
KZE 490
KZE 500
KZE 510
KZE 520
KZE 530
KZE 540
KZE 550
KZE 560
KZE 570
KZE 580
KZE 590
KZE 600
KZE 610
KZE 620
KZE 630
C THIS SECTION CALCULATES THE FUNCTIONS USING THE INTEGRAL
C REPRESENTATION, EQU 3, EVALUATED WITH 15 POINT GAUSS-
C HERMITE QUADRATURE
30 X2 = 2.000*X
Y2 = 2.000*Y
Z1 = Y2*Y2
P1 = USQR(T2*X2+M1)
Z2 = USQR(P1*X2)
T1 = EASQ(1)/(2.000*X1)
RE0 = T1*P2
IM0 = T1/P2
I1E1 = 0.0,0
I01 = 0.0,0
EU *H=2.0
T2 = X2 + TSQ(N)
P1 = USQRT(T2*2*X1+M1)
T2 = USQRT(P1+Z2)
T1 = EASQ(N)/P1
RE0 = RE0 + T1*P2
IM0 = IM0 + T1/P2
T1 = EASQ(N)*TSQ(N)
RE1 = RE1 + T1*P2
IM1 = IM1 + T1/P2
40 CONTINUE
T2 = -Y2*IM0
Z1 = Z1*H2
Z2 = Y2*IM1/H2
TERM1 = 1.4142135623730950*D(*DOSIL(Y))
TERM2 = -1.4142135623730950*D(OSIN(Y))
C THE CONSTANT IN THE PREVIOUS STATEMENTS IS OF COURSE
C SURF(Z,R) :
IM0 = RE0*TERM + T2*TERM
RE0 = RE0*TERM + T2*TERM
T1 = RE1*TERM + H2*TERM
T2 = RE2*TERM + H2*TERM
RE1 = T1*Y + T2*Y
IM1 = -T1*Y + T2*Y
RETURN
C THIS SECTION CALCULATES THE FUNCTIONS USING THE
C ASYMPTOTIC EXPANSIONS
50 TERM = 1.000
TERM = 0.000
RE0 = 1.000
IM0 = 0.000
RE1 = 1.000
IM1 = 0.000
P1 = 8.000*P2
P2 = USQRT(R2)
L = 3.910D+8.1201/P2
R1 = 1.000
P2 = 1.0
w = -b
K = J
C0 60 N=1+L
M = M + B
K = K - M
R1 = FLOAT(K-4)*M1
R2 = FLOAT(K)*M2
T1 = FLOAT(N)*P1
T2 = NTERM
RTERM = (T2*X*ITEM*Y)/T1
ITEM = (-T2*Y*TERM*X)/T1
R0 = R0 + R1*TERM
R1 = R1 + R2*TERM
R2 = R2 + R1*TERM

C THIS CONSTANT IS SQRT(P1)/2.0, WITH PI=3.14159...
1TERM = PI*DOSIN(Y)
1TERM = -P1*DOSIN(Y)
R1 = R1 + R1 + 1TERM
R2 = R2 + R2 + 1TERM
R0 = T1*M1 + T2*M2
R1 = T1*M2 + T2*M1
R2 = T1*M2 + T2*M1
M1 = T1*M2 + T2*M1
RETURN

**+**2H ARGUMENT OF THE BESSEL FUNCTIONS IS ZERO,
* 35M OR LIES IN LEFT HALF COMPLEX PLANE
END
SUBROUTINE KZEPHNE(X, Y, RE0, IM0, RE1, IM1)
LOGICAL LLLLLL
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, KU AND K1. RE0, IM0, RE1 AND IM1 GIVE
C THE REAL AND IMAGINARY PARTS OF EXP(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 FURTHER ALLOWS THIS VERSION RUNS
C ABOUT 30 PERCENT FASTER THAN ONE WRITTEN USING COMPLEX
C VARIABLES.
C DOUBLE PRECISION X, Y, X2, Y2, RE0, IM0, RE1, IM1,
* R1, K2, T1, T2, P1, P2, RTERM, ITEM, EXS(8), TSQ(8)
DATA TSQ(1)/0.00000/, TSQ(2)/0.00000/, TSQ(3)/0.00000/, TSQ(4)/0.00000/, TSQ(5)/0.00000/, TSQ(6)/0.00000/, TSQ(7)/0.00000/, TSQ(8)/0.00000/,
* /0.5641000000,0.00000/, EXS(1)/0.00000/, EXS(2)/0.00000/, EXS(3)/0.00000/, EXS(4)/0.00000/, EXS(5)/0.00000/, EXS(6)/0.00000/, EXS(7)/0.00000/, EXS(8)/0.00000/,
C THE ARRAYS TSQ AND EXS CONTAIN THE SQUARE OF THE
C ABSCISSAS AND THE WEIGHT FACTORS USED IN THE GAUSS-
C HERmite QUADRATURE.
CALL XXXXXX(0001,Z)
H2 = X*X + Y*Y
LLLLL = GT.0,0000,UK,H2,NE.0,0000
IF (LLLLL) CALL XXXXXX(0002,Z)
IF (LLLLL) GO TO 10
CALL XXXXXX(0003,Z)
WRITE (9,9999)
RETURN
10 CALL XXXXXX(0004,Z)
LLLLL = H2.GE.1.3962
IF (LLLLL) CALL XXXXXX(0005,Z)
IF (LLLLL) GO TO 20
CALL XXXXXX(0006,Z)
LLLLL = H2.GE.1.3998
IF (LLLLL) CALL XXXXXX(0007,Z)
IF (LLLLL) GO TO 30
C THIS SECTION CALCULATES THE FUNCTIONS USING THE SERIES
C EXPANSIONS
CALL XXXXXX(0008,Z)
X2 = X/2.0000
Y2 = Y/2.0000
P1 = X*Y2
P2 = Y*Y2
T1 = -(DLOG(P1+P2)/2.0000+0.577215664901532910)
C THE CONSTANT IN THE PRECEDING STATEMENT IS EULER'S
C CONSTANT
T2 = -DATAN2(Y,X)
X2 = P1 + P2
Y2 = X*Y2
RTERM = 1.0000
ITEM = 0.0000
RE0 = T1
IM0 = T2
T1 = T1 + 0.5000
HE1 = T1
IM1 = T2
P2 = COSJHT(R2)
L = 2.146000*P2 + 4.400
LLLLL=L-LL8.00-1
IF (LLLLL) CALL XXXXX(0009+2)
IF (LLLLL) CALL XXXXX(0009+2)
DO 20 N=1,L
CALL XXXXX(0011+2)
   P1 = N
   P2 = N*N
   R1 = HTERM
   HTERM = (R1*XZ-ITERM*Y2)/P2
   ITERM = (X1*Y2-ITERM*X2)/P2
   T1 = T1 + 0.50U/P1
   HE0 = HE0 + T1*TERM - T2*ITERM
   IM0 = IM0 + T1*TERM - T2*ITERM
   PI = PI + 1.00U
   TI = TI + 0.50U/P1
   H1 = H1 + (T1*TERM-T2*ITERM)/P1
   IM1 = IM1 + (T1*TERM-T2*ITERM)/P1
20 CONTINUE
CALL XXXXX(0011+2)
   M1 = H1/2 - 0.50U*(X*HE1*Y+IM1)
   H2 = -2*Y/M2 - 0.50U*(X*IM1*Y+H1)
   P1 = DEAP(X)
   HE0 = P1*HE0
   IM0 = P1*IM0
   R1 = P1*H1
   R1 = P1*P2
   RETURN
C THIS SECTION CALCULATES THE FUNCTIONS USING THE INTEGRAL
C REPRESENTATION, EQN 3; EVALUATED WITH 15 POINT GAUSS-
C WEIGHTED QUADATURE
30 CALL XXXXX(0011+2)
   AZ = 2.000*Z
   YZ = 2.000*Y
   H1 = Y2*Z2
   P1 = USHT(AZ*Z2+H1)
   P2 = USHT(P1+T2)
   T1 = EXSU(N)/P1
   HE0 = T1*P2
   IM0 = T1/P2
   HE1 = 0.000
   IM1 = 0.000
   DO 40 N=2,N
40 CONTINUE
CALL XXXXX(0011+2)
   T2 = AZ*TSU(N)
   P1 = USHT(T2*Z2+R1)
   P2 = USHT(P1+T2)
   T1 = EXSU(N)/P1
   HE0 = HE0 + T1*P2
   IM0 = IM0 + T1/P2
   T1 = EXSU(N)*TSU(N)
   R1 = R1 + T1*P2
   IM1 = IM1 + T1/P2
40 CONTINUE
CALL XXXXX(0011+2)
   T2 = -Y2*IM0
   R1 = RE1/R2
   R2 = T2*IM1/P2
   -TERM = 1.6142135623730900*DCOS(Y)
ITEM = -1.41421356237309505Y7
C THE CONSTANT IN THE PREVIOUS STATEMENTS IS OF COURSE
C SQRT(2.04)
IM0 = RE0*VIEW + IM0*ITEM
IM1 = 1.
IM2 = 1.
M = -9
K = J
DO 60 I = 1, 100
CALL XXXXXX(0011+2).
M = M + 2
K = K - M
IM0 = IM0*ITEM
IM1 = IM1*ITEM
IM2 = IM2*ITEM
CALL XXXXXX(0011+2).
END
C THE CONSTANT IS SQRT(P1)*Z0, WITH P1=3.14159...
C H1 = 1.0*TERM
C R2 = REV*TERM + IM0*TERMM
C R0 = 1.0*TERM
C IM0 = IM0*TERM
C IM1 = IM1*TERM
C IM2 = IM2*TERM
RETURN