U.S. DEPARTMENT OF COMMERCE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION
NATIONAL WEATHER SERVICE
NATIONAL METEOROLOGICAL CENTER

OFFICE NOTE 224

NWS-NMC Programming Standards

John D. Stackpole
Development Division

and

James G. Howcroft
Automation Division

NOVEMBER 1980

NWS-NMC Programming Standards

I. Philosophy

If ever there was a subject guaranteed to raise more local hackles stiffer and faster than this one, I know not what it is. No one wants to be forced into a mold; most everyone sees their own programming style as perfectly clear (to them anyway); most good programmers (i.e. those who say they enjoy working with computers and whose codes do what they are s'posed to do, most of the time) have remarkable memories for detail, and so can work over an old code with ease; in short they don't want to be bothered with a lot of fancy stuff but just want to get on with the job. Well...

There really are some reasons for adhering to a set of standards, any standards actually, just like in other fields of endeavor, such as writing. In writing you use punctuation marks, sentences, paragraphs (indented, yet), sections, (sometimes with headings), chapters, wide margins, blank lines and pages, titles, etc. All of these are arbitrary standards which, because they are generally accepted and understood, are a distinct aid in communication, author to reader. (If you think they don't make too much difference take a look at the last pages of Joyce's "Ulysses"; if your programs look like Molly Bloom's soliloquy, heaven help your associates who may have to read them).

Hey, I hear someone say, I am writing computer programs, not people programs and FORTRAN doesn't care what my codes look like. Well, yeah, but... you are not alone. Other people are going to have to read your codes, if not tomorrow, then next month, next year, when you move on, when the code has to be converted to a new machine, when modifications, additions, changes, are undertaken by others. You, two years from now, are "another person", too. Sure it takes a little extra time and trouble to make a code "readable" (surprisingly little after you get used to it, particularly if you write it "readable" from the start) but the longer term payoffs for yourself and the rest of us are substantial:

- Large Models are now or are going to be joint efforts in the future; specialists will do their thing independently of others; it will be just that much harder to talk to one another (and put the parts together) without some standardization;

- Codes that pass thru many hands can become essentially useless (even though they are operational) because the varieties of style, complexity, etc. make alterations hazardous at best. Standards will hardly cure complexity but they can make it manageable;

- Future computer conversions (they will never end) will be easier and less costly if all codes are constructed along similar lines;

- New Arrivals can be trained to "good" coding habits with a set of standards to guide them;

- Some standards, if followed closely, establish recognizable patterns in the physical and logical layout of codes: if the written codes show "non-standard" patterns, there's a good chance there is a coding or logic error in the code. This really works as a self generated de-bugging aid.

In short, standards can aid cooperative coding, clarity, consistency, communication, and conversion. And presumably save man-hours, and maybe even lead to programming that produces better codes sooner.

Now that you are all convinced, what standards? At this point nobody will agree with anyone. Thomas Jefferson understood the problem:

> And whether these forms be in all cases the most rational or not is really not of so great importance. It is much more material that there should be a rule to go by than what the rule is; that there may be a uniformity of proceeding ... not subject to the caprice ... of the members.
>
> -Jefferson's Manual, pub. 1801 (The first parliamentary procedure manual in the U. S.)

Don't knock 'em 'till you've tried 'em - they work.

II. Standards

First: Documentation.

We have constructed a documentation block template for Main programs, Subroutines, and Functions. It's sort of a fill-the-blank operation. See the attached copies. You can copy it over to your codes via the TSO QED INCLUDE command from

```
NWS.NMC.DOCBLOCK(MAIN)
 "   "      "    (SUBR)
 "   "      "    (FUNC)
```

If you do nothing else please use this documentation system. Parts of it are key words so catalogues of documentation can be compiled. We are planning to require the doc-block to be present in all NMC controlled programs (Mains, Subroutines and Functions) intended for implementation on the new computer. And for new ones on the 195's, too. We don't expect you to run about retro-fitting doc-blocks to all your old codes (it would be nice, though), but as new codes are developed or maintenance is done to old ones that show signs of a long life time, we do expect the blocks to be put in place.

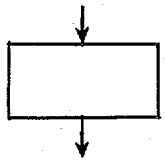
Second: Programming Style

Assertion: Structured Top-down Coding (Ah, listen to those hackles crackle...) is a good thing. See references for arguments.

Structured Rules:

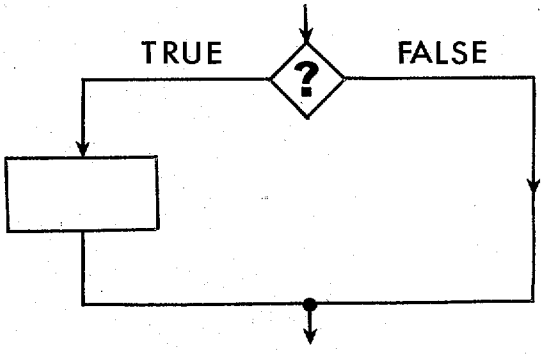
Use these basic forms ONLY (exception below)

Sequence

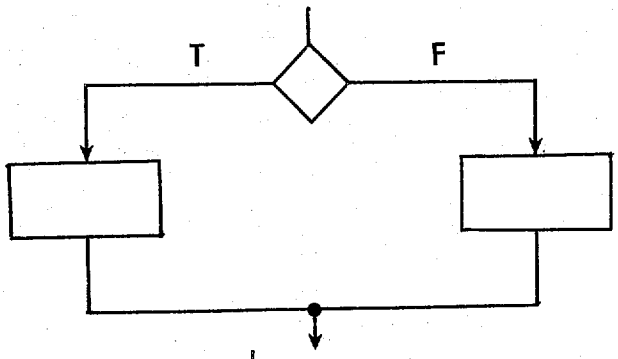


(One logical entrance to, one logical exit from, a block of instructions - no cutting back, or multiple departure routes)

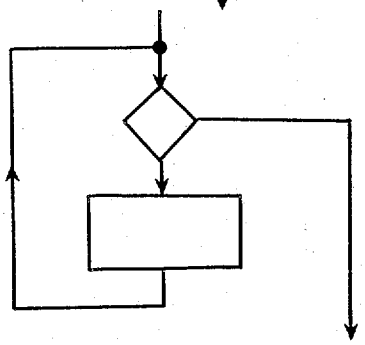
IF - THEN:



IF-THEN-ELSE:



DOWHILE



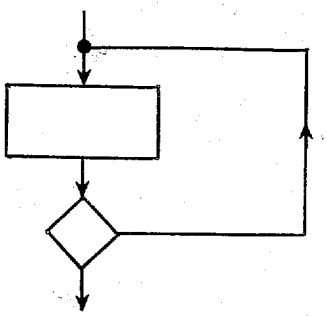
Test

Do something

The ◇ contains some condition test so the loop isn't infinite.

DO UNTIL

(a DO loop)



Do something

Test

The last two are obviously cousins.

If you don't know what these funny pictures are all about go read the IBM Structured Programming Guide.

Of course these forms can be nested; indeed they usually have to be to do what you want to do. But the nesting must be complete - each form must be finished with no cutting from one to another, out the sides so to speak. That will lead to trouble.

FORTRAN implementation.

At present there is no If...THEN...ELSE instruction in FORTRAN (FORTRAN 77 - the new one - does have "structured" instructions). Here's how to implement the forms (along with some more hackle-cracklers):

- Start your codes in Col 10 and INDENT (there, I said it) the "contents" of DO loops, IF THENs, etc. 2 (two) columns. And then keep indenting for each nested form. If you do your indenting - nesting right (and there are no logic errors) you will end up back at Col 10. If not, look for errors.

Samples:

DOUNTIL (DO-loop):

```
        DO 200  J = 1,JMAX
          A(J) = B(J)
          DO 100 I = 1,IMAX
            C(I,J) = D(I,J)
100       CONTINUE
200     CONTINUE
```

IF-THEN:

```
        IF(A.EQ.B) GOTO 300
          A   = P + 2
          AA  = Q + 3
          AAA = R + 4
300     CONTINUE
```

IF-THEN-ELSE:

```
      IF(P.EQ.Q) GOTO 600
        L = N
        Y = Z
        GOTO 700
600   CONTINUE
        L = N + 1
        Y = 5.0 * Z
700   CONTINUE
```

"Sequence" is just a series of instructions, with no GOTO type jumping about, all starting in the same column - which column depends on how many IFs and DOs have preceeded.

Note that the CONTINUEs are in the same columns as their matched DOs or IFs.

Note that structured coding is NOT "GOTO less" programming - it is just that GOTO is used ONLY in the special forms .... Except (I promised an exception) you can use GOTO in a non-structured context but only for pathological conditions (unexpected EOF, IO errors, etc.). But you should GOTO a clearly delimited Disaster Area of your code and print meaningful explanations of the trouble.

Here is a more complex example of most all the forms at once taken from real life (only the variables have been changed to protect the programmer). The outermost loop is a DOWHILE where the "test" portion involves reading a record and testing for an EOF, all at once, in one Fortran READ statement. (This is one proper exception to the "number-only-CONTINUE-statements" rule). The ERR = 9000 is a (non structured) exit to a disaster area.

```
1100    READ(NUNIT, fmt, END = 1700, ERR = 9000)  A, B, C
          F = A
          DO 1600 J = 1, JMAX
            P(J) = Q(J)
            DO 1200 I = 1, IMAX
              DIV (I,J) = DUDX (I,J) + DVDY (I,J)
              CON (I,J) = - DIV (I,J)
1200          CONTINUE
              SUMDIV = SUMDIV + DIV (1,J)
              BB(J) = A(J) * B(J)
              IF (BB(J) .LE. 0.35) GOTO 1300
                 X(J) = Y(J) + 25
                 V(J) = V(J) ** 2
                 GOTO 1500
1300          CONTINUE
                 IF (C .NE. F) GOTO 1400
                   Z(J) = A + B + C
1400             CONTINUE
1500          CONTINUE
1600        CONTINUE
            ABOUT = SUMDIV ** 2
            GOTO 1100
1700    CONTINUE
```

[if... then] [if... then else] [DOWHILE]

(You might meditate on how this block of code would look if every line began in the same column.)

Sundries:

Here's a collection of dos and don'ts which in our experience help greatly in doing all the good things we want to do ...

- Each DO, IF, or GOTO shall have its <u>own</u> unique CONTINUE - paired one for one.
- Only CONTINUE and FORMAT should have statement numbers and only one reference per CONTINUE.
- In the specification section <u>order</u> and <u>group</u> things thusly (Start in Col 10)

```
          REAL*8    .............
     1              .............
C
          REAL*4    .............
     1              .............
C
          INTEGER   .............
C
          LOGICAL   .............
```

This is not an exhaustive list - the idea is to put the larger word variables first and go down to the smallest like LOGICAL*1.

Variables and Continuations should start in Col 20 or so - try to line up variables vertically. Separate each block of type declarations with a blank Comment or two.

- Do NOT use "DIMENSION"
  Use INTEGER*4, REAL*8 etc.

- Do not use blank COMMON - label your COMMONs.

- There should be no numbers (except 1 and statement numbers) in the body of the code, including DO loop limits. Group all number constants at top of code in specification (DATA) statements or a series of initial instructions. Comment explain unusual ones like unit conversion factors. Give units of physical constants, (or use SI throughout). This rule includes Fortran unit numbers. Make them (INTEGER*4) variables too, and specify them up front. This will make subsequent adjustments to operational frameworks (e.g. MEGASTEP) much easier.

- Make Statement numbers (initially) sequential (by 100s).

- Avoid the arithmetic IF completely - use logical IFs and nest (and indent) as necessary.

- Invent meaningful variable names that describe themselves.

- Do not use "Computed GOTO" -
  Use sequence of

```
IF(  ) ....
IF(  ) ....
IF(  ) ....
```

  (This is an IF...ELSE IF...ELSE IF...ELSE form, a variant of nested IF THENs)

- Put blank Comment spacers between logical subunits of the code, and comments explaining the tricky parts (as well as Unit title comments).

- Do not use ENTRY in subroutines.

- Try (this is a tough one) to keep any code (Main or any Subroutine) to no more than 100-150 lines of instructions (excluding comments). Things bigger than this are hard to grasp - parts are out of sight, causing errors to be more easy to make. The routine should form a logical unit. Modularize.

A final (at last) word:

All this looks like a lot but it really isn't. The pay off is substantial - if nothing else you end up with a code that looks nice, really. It's esthetically pleasing, not foreboding, and something you can take pride in.

A couple of last last words:

Again we do not expect you to go back and structure all your old codes. But have at it with new ones. Sure, it will take you longer in the beginning, but don't give up. Also if you are putting patches in old codes do the patches by the rules. That will look funny but at least you will be able to tell what's new. The idea though is to think and use structure for all your coding from now on - after a while you will look with considerable disdain on your earlier non-structured efforts (and especially those of your associates).

Ultimate final urging:

Do try the indenting business - I know it looks weird at first (those staircases of CONTINUEs especially) but a surprising lot of self-debugging error detection comes out of this. What happens is that the logical structure is shown, matched, or manifested by the physical structure of the code. When the two structures don't come out the same (i.e. the bottom CONTINUE in Col 10) then you know there is an error somewhere. That knowledge is a lot of power. You don't suspect it, you know it. All you have to do is find it.

And it sure does help readability.

FURTHER READING

1. IBM FORTRAN Manual

   No kidding - go back and re-read the manual: you will be surprised at how much you have forgotten or not realized (first time through) how useful some things are.

2. IBM Structured Programming Guide
   (Copy available in Systems Evaluation Branch).

   Good meat-and-potatoes exposition of how and why.

3. The Elements of Programming Style
   Kernighan & Plauger. McGraw Hill N Y 1973 (Paperback).

   Absolutely outstanding. A must for any and all programmers.

4. FORTRAN with Style: Programming Proverbs.
   Ledgard & Chmura. Hayden, N. J. 1978.

   More good advice (with good examples) although a little cutesie at times.

5. Top Down Structured Programming Techniques.
   McGowan & Kelly Petrocelli/Charter N. Y. 1975.

   Good intro to theory (and practice).

6. Structured Programming.
   Linger, Mills & Witt Addison Wesley Reading Mass 1979.

   Graduate Level theory - heavy stuff.

There are lots of other books on the subject but they get repetative after a while. If you can read only one, read Kernighan & Plauger's Elements.

```
C$$$  MAIN PROGRAM DOCUMENTATION BLOCK  ***                                    00000010
C                                                                              00000020
C MAIN PROGRAM: NAMEXXXX  DESCRIPTIVE TITLE (40 CHARACTERS)                    00000030
C   AUTHOR:                        ORG:                    DATE: DD MON YY     00000040
C                                                                              00000050
C ABSTRACT:  (START ABSTRACT HERE AND INDENT TO COLUMN 5 ON THE                00000060
C   FOLLOWING LINES, FOR AS MANY LINES AS ARE REQUIRED TO PROVIDE              00000070
C   A REASONABLE DESCRIPTION OF THE CODE'S PURPOSE.)                           00000080
C                                                                              00000090
C                                                                              00000100
C                                                                              00000110
C USAGE:                                                                       00000120
C                                                                              00000130
C   INPUT FILES : (NAMES & USAGE) (PARM, IF INCLUDED)                          00000140
C     FTNNF001 (START IN COL 7)                                                00000150
C     FXN                                                                      00000160
C                                                                              00000170
C   OUTPUT FILES: (NAMES & USAGE)                                              00000180
C     FTMMF001 (START IN COL 7)                                                00000190
C                                                                              00000200
C                                                                              00000210
C   SUBPROGRAMS CALLED:                                                        00000220
C     UNIQUE : (LIST ALPHABETICALLY HERE)                                      00000230
C                                                                              00000240
C     LIBRARY: (DITTO)                                                         00000250
C                                                                              00000260
C   EXIT STATES:                                                               00000270
C                                                                              00000280
C     COND = 0   SUCCESSFUL RUN                                                00000290
C          = N   TROUBLE (SPECIFY WHATEVER)                                    00000300
C          = NN  BIG TROUBLE (SPECIFY WHAT)                                    00000310
C                                                                              00000320
C   REMARKS:  LIST CAVEATS, ADD HELPFUL HINTS OR INFORMATION IN THIS           00000330
C     SPACE THAT WOULD BE USEFUL TO SOMEONE INTERESTED IN THE PROGRAM.         00000340
C                                                                              00000350
C ATTRIBUTES:                                                                  00000360
C   LANGUAGE:   (INCLUDE VENDOR EXTENSIONS USED)                               00000370
C   SOURCE STATEMENTS:   (NUMBER OF)        PGM SIZE:   (BYTES)                00000380
C                                                                              00000390
C$$$                                                                           00000400
C                                                                              00000410
```

```
C$$$  SUBROUTINE DOCUMENTATION BLOCK  ***                                  00000010
C                                                                          00000020
C SUBROUTINE: NAMEXXXX  DESCRIPTIVE TITLE (40 CHARACTER LIMIT)             00000030
C    AUTHOR:                      ORG:                  DATE: DD MON YY    00000040
C                                                                          00000050
C ABSTRACT:  (START ABSTRACT HERE AND INDENT TO COLUMN 5 ON THE            00000060
C    FOLLOWING LINES, FOR AS MANY LINES AS ARE REQUIRED TO PROVIDE         00000070
C    A REASONABLE DESCRIPTION OF THE CODE'S PURPOSE.)                      00000080
C                                                                          00000090
C                                                                          00000100
C                                                                          00000110
C USAGE:  CALL NAME (ARGUMENTS)                                            00000120
C    INPUT ARGUMENTS:  (CONTENTS AND USAGE)                                00000122
C                                                                          00000130
C    INPUT FILES:  (IF ANY)                                                00000140
C                                                                          00000150
C                                                                          00000160
C    OUTPUT ARGUMENTS:                                                     00000172
C                                                                          00000174
C    OUTPUT FILES: (NAMES & USAGE)                                         00000180
C                                                                          00000190
C                                                                          00000200
C    RETURN CONDITIONS:                                                    00000210
C                                                                          00000212
C    SUBPROGRAMS CALLED:                                                   00000220
C      UNIQUE :  (LIST ALPHABETICALLY HERE)                                00000230
C                                                                          00000240
C      LIBRARY:  (DITTO)                                                   00000250
C                                                                          00000260
C ATTRIBUTES:                                                              00000360
C    LANGUAGE:   (INCLUDE VENDOR EXTENSIONS USED)                          00000370
C    SOURCE STATEMENTS:   (NUMBER OF)        PGM SIZE:    (BYTES)         00000380
C                                                                          00000390
C$$$                                                                       00000400
C                                                                          00000410
```

```
C$$$  FUNCTION DOCUMENTATION BLOCK  ***                                  00000010
C                                                                        00000020
C FUNCTION: NAMEXXXX  DESCRIPTIVE TITLE (40 CHARACTER LIMIT)             00000030
C   AUTHOR:                      ORG:                  DATE: DD MON YY   00000040
C                                                                        00000050
C ABSTRACT:  (START ABSTRACT HERE AND INDENT TO COLUMN 5 ON THE          00000060
C   FOLLOWING LINES, FOR AS MANY LINES AS ARE REQUIRED TO PROVIDE        00000070
C   A REASONABLE DESCRIPTION OF THE CODE'S PURPOSE.)                     00000080
C                                                                        00000090
C                                                                        00000100
C                                                                        00000110
C USAGE:  XX = NAME (ARGUMENTS)                                          00000120
C   INPUT ARGUMENTS:  (CONTENTS AND USAGE)                               00000122
C                                                                        00000130
C   INPUT FILES:  (IF ANY)                                               00000140
C                                                                        00000150
C                                                                        00000160
C   OUTPUT ARGUMENTS:                                                    00000172
C                                                                        00000174
C   OUTPUT FILES: (NAMES & USAGE)                                        00000180
C                                                                        00000190
C                                                                        00000200
C   RETURN CONDITIONS:                                                   00000210
C                                                                        00000212
C   SUBPROGRAMS CALLED:                                                  00000220
C     UNIQUE :  (LIST ALPHABETICALLY HERE)                               00000230
C                                                                        00000240
C     LIBRARY:  (DITTO)                                                  00000250
C                                                                        00000260
C ATTRIBUTES:                                                            00000360
C   LANGUAGE:   (INCLUDE VENDOR EXTENSIONS USED)                         00000370
C   SOURCE STATEMENTS:   (NUMBER OF)        PGM SIZE:   (BYTES)          00000380
C                                                                        00000390
C$$$                                                                     00000400
C                                                                        00000410
```

U.S. DEPARTMENT OF COMMERCE
National Oceanic and Atmospheric Administration
NATIONAL WEATHER SERVICE
National Meteorological Center

OA/W324/JDS

November 5, 1980

TO : All NMC Personnel With Anything to Do With Computer Programs

FROM : OA/W3 - Frederick G. Shuman FGShuman

SUBJECT: Standards for Programs

The attached Office Note #224 contains a set of NMC programming standards which you should follow in your future programming efforts, starting now. In particular, the use of the "Documentation Block" (see Page 2) is mandatory for all individual programs (MAIN, SUBROUTINE, or FUNCTION) which are intended to be run operationally on the new computer, for new operational codes for the 360/195, and for currently run operational 360/195 programs when substantial revisions or maintenance is done to them.

The Implementation Committee will review new codes against these standards, and require the programmer to "retrofit" his code for implementation.

Attachment