**NATIONAL WEATHER SERVICE**
**OFFICE of HYDROLOGIC DEVELOPMENT**

# Science Infusion Software Engineering Process Group (SISEPG) FORTRAN Programming Standards and Guidelines

**Version 1.6**

## Revision History

| **Date** | **Version** | **Description** | **Author** |
|---|---|---|---|
| 02/17/2005 | 1.0 | Initial Version | Scott Vandemark |
| 03/31/2006 | 1.1 | Edit & Reformat | Marylin Andre |
| 04/14/2006 | 1.2 | Incorporated comments from SISEPG group | Cham Pham |
| 04/20/2006 | 1.3 | Change font on examples | M. Andre |
| 05/02/2006 | 1.4 | Additional revisions | M. Andre |
| 06/29/2006 | 1.5 | Apply team member revisions | M. Andre |
| 08/03/2006 | 1.6 | Apply HSEB Chief comments – remove reference to FORTRAN 90 standard of 31 characters | M. Andre |

## Table of Contents

Revision History ........ i
Table of Contents ........ ii
1. Introduction ........ 1
2. General Information ........ 2
    2.1 Standards ........ 2
    2.2 Guidelines ........ 2
3. Non-Executable Statements ........ 5
    3.1 Standards ........ 5
    3.2 Guidelines ........ 6
4. Executable Statements ........ 7
    4.1. Standards ........ 7
    4.2. Guidelines ........ 8

# 1. Introduction

The Office of Hydrologic Development (OHD) produces software which NWS Weather Forecast Offices (WFOs) and River Forecast Centers (RFCs) use to create hydrologic forecasts for rivers and streams across the country. Just like many other organizations, software has become a critical component supporting the operations of these forecast offices. Because software plays such an important role, it is essential that it be well-written and maintained.

The OHD Science Infusion Software Engineering Process Group (SISEPG) is developing standards and guidelines to ensure that programmers follow good, widely accepted software development practices when coding. It is believed that this will lead to well-written and better structured programs.

Well-written software offers many advantages. It should contain fewer bugs and run more efficiently than poorly written programs. It also makes it easier for a programmer who was not involved in the development of the software to learn how it works.

Software has a life-cycle. A large part of its life-cycle revolves around maintenance. Software may exist for many years, even decades. Long after the original programmer has moved on, the software will require maintenance in the form of bug fixes and enhancements. The time spent doing this and hence the cost is greatly reduced when the code is developed and maintained according to software standards.

The FORTRAN programming language is a popular and powerful application memory and system routines which if used improperly can result in unreliable programs which waste system resources and CPU cycles.

This document presents standards and guidelines for the FORTRAN Programming Language. The standards are programming techniques which OHD programmers are expected to follow. Their use will be enforced through peer reviews and code walkthroughs. The programming guidelines are good programming practices which developers are encouraged to adopt.

FORTRAN programming standards are a set of FORTRAN programming rules which must be applied by FORTRAN developers when creating programs. These techniques are considered best practices which greatly enhance the readability and maintainability of a program. During a code walkthrough, the software reviewers will be inspecting the code to make sure that these standards are followed.

FORTRAN programming guidelines are a set of FORTRAN programming practices which are considered best practices which greatly enhance the readability and maintainability of a program. While developers are not required to use these techniques, they are encouraged to integrate them into their programming style.

Programmers should read the OHD Software Development Standards and Guidelines document to become familiar with the standards and guidelines deemed by the SISEPG to be applicable to all programming languages.

## 2. General Information

### 2.1 Standards

- See the OHD General Software Development Standards and Guidelines document for standards on writing prologue documentation in a source file.
- See the OHD General Software Development Standards and Guidelines document for standards on how to make files more readable and maintainable.
- Use standard FORTRAN 77. Do not use compiler dependent features.
- FORTRAN 77 data names must consist of one to six characters. Use a letter as the first character and only letters or numerals for the remaining 5 possible characters.

### 2.2 Guidelines

- See the OHD General Software Development Standards and Guidelines document for guidelines on how to make files more readable and maintainable.
- The names of source files which belong to a common library or an executable should have a common prefix which identifies them as being part of that library or executable. This also applies to INCLUDE files. This helps other programs quickly determine which library a source file belongs to.

  Example:

  - Source files
  ```
  pdc_engine_parser.f
  pdc_engine_reader.f
  pdc_engine_writer.f
  ```

  - Include file
  ```
  pdc_engine_parser.inc
  ```

- Within a program module that has a group of subprograms the subprograms should be in alphabetical order.
- Use consistent indentation. See the OHD General Software Development Standards and Guidelines document for guidelines.

  Example:

  Bad:

  ```
  C THIS EXAMPLE DOES NOT USE ANY INDENTATION.
  ```

```
    PROGRAM MAIN
    PARAMETER (MVAL=10)
    DIMENSION IVAL(MVAL)
    DO 10 I=1,MVAL
C CHECK THE VALUE OF I
    IF (I.LE.5) IVAL(I)=0
    IF (I.GT.5) IVAL(I)=1
10  CONTINUE
    DO 20 I=1,MVAL
    IF (IVAL(I).EQ.0) THEN
C THE VALUE OF I IS 0
    IVAL(I)=1
    WRITE (6,*) 'IVAL CHANGED FROM 0 TO 1'
    ENDIF
    IF (IVAL(I).EQ.1) THEN
C THE VALUE OF I IS 1
    IVAL(I)=0
    WRITE (6,*) 'IVAL CHANGED FROM 1 TO 0'
    ENDIF
20  CONTINUE
    STOP
    END
```

Better:

```
C  THIS EXAMPLE USES CONSISTENT INDENTATION.
   PROGRAM MAIN
   PARAMETER (MVAL = 10)
   DIMENSION IVAL(MVAL)

    DO 10 I = 1, MVAL
C   CHECK THE VALUE OF I
       IF (I .LE. 5) IVAL(I) = 0
       IF (I .GT. 5) IVAL(I) = 1
10  CONTINUE

    DO 20 I = 1, MVAL
       IF (IVAL(I) .EQ. 0) THEN
C   THE VALUE OF I IS 0
           IVAL(I) = 1
           WRITE (6,*) 'IVAL CHANGED FROM 0 TO 1'
       ENDIF

       IF (IVAL(I) .EQ. 1) THEN
C   THE VALUE OF I IS 1
           IVAL(I) = 0
           WRITE (6,*) 'IVAL CHANGED FROM 1 TO 0'
       ENDIF
20  CONTINUE

    STOP
    END
```

- Do not use inline comments following an exclamation mark (!) as is allowed by some compilers.

- Avoid combining variable initialization with type declarations in one statement as is allowed by some compilers.
- Do not write more than one statement per line as is allowed by some compilers.

  Example:

  Bad:

  ```
  IPSPAG(LP) = 0; CALL UPAGE(LP)
  ```

  Better:

  ```
  IPSPAG(LP) = 0
  CALL UPAGE(LP)
  ```

- Use only those intrinsic functions which appear in the standard (and are indicated as such in most compiler manuals). Do not call host system services directly.
- Use only the standard character set.  Avoid lower case characters. Use only the following special characters:

  ```
  blank, . ' : = + * / ( ) $
  ```

- Use only the apostrophe (') to delimit character strings.
- Main programs should always begin with the statement PROGRAM [name] which should have no associated parameter list.
- Use meaningful names.
- Do not use FORTRAN keywords as symbolic names.
- Initialize all variables. Do not assume machine default value assignments. Do not initialize variables of one type with values of another.
- Do not split a name or keyword between two lines.

  Example:

  Bad:

  ```
    IF (SWORK(IDX).GT.SWORK(IDY)-0.001.AND.SWORK(IDX).LT.S
  *     WORK(IDY)+0.001) THEN
  ```

  Better:

  ```
    IF (SWORK(IDX) .GT. SWORK(IDY)-0.001 .AND.
  *     SWORK(IDX) .LT. SWORK(IDY)+0.001) THEN
  ```

- If a statement consists of an initial line and one or more continuation lines then the lines should be split in a logical manner.

Example:

Bad:
```
 IF (SWORK(IDX).GT.SWORK(IDY)-0.001.AND.SWORK(IDX).LT.SWORK
*     (IDY)+0.001) THEN
```

Better:
```
 IF (SWORK(IDX) .GT. SWORK(IDY)-0.001 .AND.
*    SWORK(IDX) .LT. SWORK(IDY)+0.001) THEN
```

- Check input for errors. An error code should be set that is returned to the calling routine. Errors should be indicated by printing an error message. An error handling routine can also be called.

- Whenever possible the value of the 'no error' condition return code variable should be zero. The variable should be INTEGER and not LOGICAL or some other type.
- Printed output should be arranged in an easy to read format. For example, line up columns of numbers. Also identify values printed in clear terms or in terms of variables defined in the program description.
- Main programs should print a successful completion message such as "PROGRAM XXX COMPLETED" where XXX is the program name.  Any other stop should produce a print message that indicates the problem and where the stop occurred.  If the stop is in a subroutine then an error statement which includes the subroutine name should be printed.

# 3. Non-Executable Statements

## *3.1 Standards*

- Explicitly declare all INTEGER and REAL variables, constants or functions. Do not use the FORTRAN predefined specification of:
  - `integer begins with I-N`
  - `real begins with A-H and O-Z`
- Avoid using the EQUIVALENCE statement unless there is no other alternative way to structure the program.
- Group FORMAT statements at the end of each subprogram.

  or

- Group FORMAT statements in the code where they are referenced and number them in sequence along with other numbered lines. Put a FORMAT statement following the first I/O statement which refers to it.

## 3.2 *Guidelines*

- Use only the standard explicit types. Avoid reduced and extended precision as well as octal, hexadecimal and Hollerith unless essential for the application. The only standard explicit types are REAL, INTEGER (no size specification), LOGICAL, CHARACTER, DOUBLE PRECISION and COMPLEX.

  Example: - `Use DOUBLE PRECISION instead of REAL*`**`8`**

  `- Use INTEGER instead of INTEGER*`**`4.`**

- Declaration and DATA statements should immediately follow the documentation block in logical order. An order such as PARAMETER, variable type, DIMENSION, COMMON, and DATA is appropriate. Always use PARAMETER first and DATA last.
- COMMON declarations and INCLUDE statements should follow the declaration and DATA statements and be in alphabetical order. An array in a COMMON block should have its dimensions declared in the COMMON statement.
- Ensure that each COMMON block is defined the same in all subprograms; All COMMON block names should be different from all subprogram names.
- Do not pass as an argument any variable referenced in a COMMON block in both the calling and called subprogram.
- Avoid mixing variables of type CHARACTER with variables of other types in COMMON blocks.
- Avoid using a BLOCK DATA subroutine when DATA statements are used for initialization of COMMON block variables. The method by which BLOCK DATA is loaded is system dependent and must be understood on a given processor. Some systems require explicit linking of the BLOCK DATA subroutine when used in a library.
- If the dimensions of a variable may be changed, and always when the dimensions are referred to in other statements (e.g. to keep from overflowing the array), the dimensions should be declared by defining a variable in a PARAMETER statement. The actual number should never be referred to in the code but rather referred to by the variable name used in the PARAMETER statement.

  Example:

  Bad:

```
      DIMENSION IVAL(10)
      DO 10 I = 1, 10
         IVAL(I) = 0
 10   CONTINUE
```

  Better:

```
   PARAMETER (MVAL = 10)
```

```
      DIMENSION IVAL(MVAL)

      DO 10 I = 1, MVAL
         IVAL(I) = 0
  10  CONTINUE
```

- Use a DATA statement to assign initial values to variables. Arrange DATA statements so that they can be easily read. Put on separate lines values pertaining to different dimensions.
- Array subscript expressions should be of type INTEGER only.
- Array references should always contain the same number of subscripts as in the array declaration and should not assume values outside the lower and upper bounds of the declared dimensions.
- If a FORMAT specification is stored in an array then the array should be of type CHARACTER. FORMAT specifiers should be separated by a comma.

# 4. Executable Statements

## *4.1. Standards*

- Terminate all subprograms by a RETURN and an END statement.
- Avoid the use of multiple STOP statements.
- Use a logical scheme for indicating continuation lines such as a sequence of numbers 1 through 9 then alphabetical order starting with the letter A or the same character for all continuation lines.
- Use a logical scheme for a subroutine call sequence. A suggested order is:

  a. logical unit number variables
  b. other input variables
  c. variables used for both input and output or work space
  d. output variables ending with an error (return) code

  Example:

```
 PROGRAM MAIN
IPR = 6
INVAL = 1
CALL SUB1 (IPR, INVAL, ISTAT)
WRITE (IPR,*) 'RETURN 1 ENCOUNTERED IN SUB1'
WRITE (IPR,*) 'ISTAT=', ISTAT
STOP
END

SUBROUTINE SUB1 (IPR, INVAL, ISTAT)
```

```
      ISTAT = 0
      WRITE (IPR,*) 'IN ROUTINE SUB1'
      IF (INVAL .EQ. 1) ISTAT = 1
      RETURN
      END
```

- Do not use any other source column format than the standard one:

  a. 1-5: label

  b. 6: continuation

  c. 7-72: statement

## 4.2. Guidelines

- Do not use more than 19 continuation lines for any one line of code.
- Do not use any nonblank characters in columns 1 through 5 of a continuation line.
- Avoid using any alternate return symbols (indicated by an asterisk in the SUBROUTINE statement).
- Each subroutine should have only one entry point. Do not use the ENTRY statement.
- Do not define external functions having the same name as intrinsic functions.
- Use the asterisk (*) notation to declare the last dimension an array passed as arguments if the extent in that dimension is unknown to the called routine.
- Use the asterisk (*) notation to declare the length of CHARACTER variables passed as arguments if their length is unknown to the called routine.
- Statement functions should be contained between comment lines in a manner which makes clear that they are not the first executable statements.
- Statement numbers should be in ascending order and left-adjusted.
- A statement number should occur only on a CONTINUE or FORMAT statement.
- DO loop control parameters should be of type INTEGER only.
- Terminate each DO loop on a separate CONTINUE statement and indent the body of the loop.
- Do not pass control into a DO loop, IF block, or ELSE clause other than by the normal initial statement.
- Avoid abbreviations for .TRUE. and .FALSE.
- Avoid using control statements (e.g. Unconditional transfer (GO TO statement), the logic IF, the arithmetic IF, computed GO TO, and assigned GO TO implementation versions of the case statement) unless no other way can be found

to structure the program.

Example:

```
- Arithmetic IF statement:  IF (expression) 10, 20, 30
- Logical IF statement:     IF (expression) GO TO 10
- Assigned GO TO:           ASSIGN 20 TO IVAR
                            GO TO IVAR, (10, 20, 30)
                               .........
                       20      CONTINUE
```

- Use parentheses to control evaluation order in expressions.
- Do not compare arithmetic expressions of different types.
- I/O statements should contain the parameters END= and IOSTAT= as appropriate.
- Do not make use of the value of the IOSTAT parameter because it may be system dependent. Use only its sign.

  Example:

  Bad:

```
   READ (ICD,*,IOSTAT=IOSTAT)
   IF (IOSTAT .EQ. 900) WRITE (IPR,*) 'IOSTAT 900
*  ENCOUNTERED'
```

  Better:

```
   READ (ICD,*,IOSTAT=IOSTAT)
   IF (IOSTAT .GT. 0) WRITE (IPR,*) 'IOSTAT', IOSTAT,
*  ‘ENCOUNTERED'
```

- Avoid using the parameter ERR= on I/O statements.
- Use WRITE rather than PRINT statements for nonterminal I/O.
- Logical unit numbers in I/O statements should be an INTEGER variable. Avoid using an asterisk (*). The variable should be given a value in the MAIN program and passed to subroutines through the argument list or in a COMMON block.
- Do not use nonstandard I/O statements such as ENCODE or DECODE (use internal files), DEFINE FILE (use OPEN), NAMELIST, PUNCH, buffer in/buffer out and other asynchronous operations.
- An array with multiple dimensions should be addressed such that the first indices vary the fastest and the last indices vary the slowest. This can avoid dramatic losses in efficiency on certain computers under certain conditions. Multi-dimensional data array is stored in memory using the column-major order.

  Example:

```
   INTEGER ROW, COL, I, J
   PARAMETER (ROW = 5, COL = 3)
```

```
      INTEGER IVAL(COL, ROW)
      DO 20 J = 1, ROW
         DO 10 I = 1, COL
            IVAL(I,J) = 0
   10    CONTINUE
   20 CONTINUE
```

## 5. References

1. A summary of Chapters 8 and 9 of the book Effective FORTRAN 77, Michael Metcalf, Oxford University Press, 1985, ISBN 0198537093 found at the Web site http://dbwww.essc.psu.edu/lasdoc/programmer/4fortran.html

2. The National Weather Service Meteorological Development Laboratory FORTRAN standards found at the Web site http://www.mdl.nws.noaa.gov/~qa/pdf_files/MDL_FORTRAN_Standards.pdf