
SPSRB Common Standards Group

Fortran 77 Programming Standards and Guidelines

Kenneth A. Jensen

Raytheon Information Solutions



Version 1.0

April, 2009

1. INTRODUCTION

The objective of this document is to provide standards for Fortran 77 code that is intended for operational use through the SPSRB process. The intended users of this document are programmers, testers, and reviewers of Fortran 77 code that will be used to implement an algorithm that creates an operational product from remote sensing satellite data. To achieve the objective, this document shall:

- Establish Fortran 77 programming standards, drawn from international standards
- Provide Fortran 77 programming guidelines
- Provide examples of good Fortran 77 programming practices

This document is based on the recognition that many Fortran programmers and potential reviewers are accustomed to the Fortran 77 version, and much of the legacy Fortran code is in the Fortran 77 style. While it is recommended that Fortran programming be in accordance with Fortran 90 or later standards when practical, it is not required. Fortran 77 code, when written according to its standards, is compatible with newer Fortran compilers.

2. REFERENCE DOCUMENTS

ISO 1539:1980 is the international standard for Fortran 77 code. This is a very large document that can be used as a reference at the programmer's discretion. Equivalent information can be found on-line at http://www.fortran.com/fortran/F77_std/rjcnf0001.html.

Transition from Fortran 77 to Fortran 90 is a useful training document for Fortran 77 programmers who are interested in programming with Fortran 90 and later versions of Fortran. This document is available on the SPSRB website.

Boukabara, S.-A. and P. van Delst (2007), *Standards, Guidelines and Recommendations for Writing FORTRAN 95 Codes* provides the NOAA/NESDIS common standards, guidelines and recommendations for Fortran 95 code. This document is available on the SPSRB website.

3. DEFINITIONS

Main Program. A Fortran main program begins with the reserved word PROGRAM and ends with a matching reserved word END, and consists of a sequence of executable statements and optional declarations. A program is always a separately compilable unit.

Program Unit. A program unit is any of the three structures in Fortran, namely PROGRAM, SUBROUTINE, and FUNCTION.

Subprogram. A Fortran subprogram begins with either the reserved word FUNCTION, SUBROUTINE, MODULE, or BLOCK DATA and ends with a matching reserved word END. It consists of a sequence of executable statements and is called by a main program or by another subprogram. A subprogram may or may not be a separately compilable unit, depending upon implementation.

4. PROGRAMMING STANDARDS AND GUIDELINES

This section contains the SPSRB Fortran programming standards and associated guidelines.

It is recognized in the spirit of this standard that certain suggestions which make code easier to read for some people (e.g. lining up attributes, or using all lower case or mixed case) are subjective and therefore should not have the same weight as techniques and practices that are known to improve code quality. For this reason, the standards within this document are divided into three components; Standards, Guidelines and Recommendations:

- *Standards:* Aimed at ensuring portability, readability and robustness. Compliance with this category is mandatory. Identified by the (***) sign.
- *Guidelines:* Good practices. Compliance with this category is strongly encouraged. The case for deviations will need to be argued by the programmer. Identified by the (**) sign.
- *Recommendations:* Compliance with this category is optional, but is encouraged for consistency purposes. Identified by the (*) sign.

4.1. Language Features

Standards:

(***) Only language features and capabilities that are documented or defined in the **ISO 1539:1980 (c.f. Section 2)** shall be used. Use the -iso compiler flag to ensure this.

(***) Fixed source form (Fortran 77) and free source form (Fortran 90 ¹) shall not be mixed within a subprogram. Program units written in fixed form and free form may be mixed in the same program, but each unit must be only in one form and at the compilation both forms may not be in the same source file.

Guidelines:

(**) If an algorithm is re-using a substantial amount of legacy Fortran 77 code, complete the code in the Fortran 77 (fixed form) style. If there is no re-use, new code should be re-written in free form style. If the amount of re-use is small, new code should be written in free form style and legacy code re-written in free form style. An exception to this guideline is the case where the designated programmers are not familiar with free form style and the scope of the project does not warrant the training cost. In this case, new code written in fixed form style is acceptable.

Recommendations:

(*) During the development and testing phases, use compiler options that provide additional checks of the code.

¹ In this document, references to Fortran 90 are intended to apply to all later versions on Fortran (e.g. Fortran 95).

4.2. Readability

Standards:

(***) When a single statement extends to more than one (1) line, begin each continuation line with an ampersand (&) in column six (6), and indent two (2) levels of indentation relative to the first statement line.

(***) Begin each continuation line with an ampersand (&) in column six (6), and indent two (2) levels of indentation relative to the first statement line.

4.9 Error Trapping

Standards:

(***) Take special care with I/O statements since these are usually affected by events beyond the control of the programmer. Include an item of the form ERR=label which causes control to be transferred to the statement attached to that label in the event of an error. This must, of course, be an executable statement and in the same program unit. For example:

```
    READ(UNIT=IN, FMT=*, ERR=999) VOLTS, AMPS
    WATTS = VOLTS * AMPS
    rest of program in here . . . . . and finally
    STOP
    WRITE(UNIT=*,FMT=*)'Error reading VOLTS or AMPS'
    END
```

(***) Handle the end-of-file condition when reading beyond the end of a sequential or internal file. If an item of the form: END=label (as opposed to ERR=999 in the above example) then control is transferred to the labeled statement when the end-of-file condition is detected. The END= keyword may only be used in READ statements, but it can be used in the presence of both ERR= and IOSTAT= keywords. End-of-file detection is very useful when reading a file of unknown length.

4.10 Statement Numbers

Standards:

(***) Consistently justify statement numbers in columns two (2) through five (5) in ascending order throughout a program.

Guidelines:

(**) Avoid referencing statement numbers in comments.

4.11 Subroutine Control

Standards:

(***) Do not use an alternate return specifier as an argument in a calling sequence in the event of an error (e.g., "CALL foo (a, b, *999)").

4.12 Statements

Standards:

(***) All variables shall be declared using a type-statement, INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, and CHARACTER.

(***) If COMMON blocks are used, the following practices should be followed:

- All variables in a COMMON block should be named the same in every program unit that uses the common data.
- COMMON blocks shall be declared in a separate file and copied into the source file using the INCLUDE statement.
- Variables in the COMMON statement shall be aligned on their proper word and byte boundaries. This generally means listing the variables in descending order according to their data type size: quad-word variables first, then double-word variables, word variables, half-word variables, and finally single byte variables.

(***) The DO statement shall not contain any statements that change the value of the loop-controlling variable.

(***) All loops shall terminate with a unique CONTINUE statement.

(***) Every GO TO statement shall be accompanied by comments placed near the GO TO statement to document the applicable constraints and comments placed near the statement receiving control to document the origin of the transfer of control.

(***) GO TO statements shall not be used to transfer control outside loops.

(***) Each GO TO statement shall target a unique CONTINUE statement.

(***) Assigned GO TO statements shall not be used.

Guidelines:

(**) Computed GO TO or ELSE IF statements are acceptable in lieu of a CASE statement. However, out of range conditions should be included in the statement.

(**) The use of GO TO statements in new Fortran 77 code is discouraged. GO TO statements should be used only where required to meet specific execution time, space constraints, or to reduce unnecessary complexity.

Recommendations:

(*) Use global variables or modules instead of COMMON blocks and INCLUDE files.

4.12.1 IMPLICIT NONE Statement

Standards:

(***) All program units shall include the "IMPLICIT NONE" statement and should be compiled with the option so the compiler flags any variables that are not explicitly declared.

4.12.2 DO Statement

Standards:

(***) Indent the statements following the Fortran DO statement one (1) level of indentation. Format the DO statement as follows:

```
                DO <stmt#> <expression>
                   statements
<stmt#>         CONTINUE
```

4.12.3 FORMAT Statement

Standards:

(***) Accompany each READ and WRITE statement with the corresponding FORMAT statement. A format statement must be coded only once and be placed after the first I/O statement that refers to it. Example:

```
                READ (5,10) SIZE
10  FORMAT(I5)

                READ (5,10) SIZE2
```

4.12.4 IF Statement

Standards:

(***) Enclose the condition(s) following the Fortran reserved word IF in parentheses.

(***) Indent the statements following the IF, ELSE IF, and ELSE statement, one (1) level of indentation.

(***) Format the IF statement as follows:

```
IF ( <expression> ) THEN
  statements
END IF
```

(***) Format the Arithmetic IF statement as follows:

```
IF ( <expression> ) s#1, s#2, s#3
C
s#1  CONTINUE
    <statements>
    GO TO s#4
C
s#2  CONTINUE
    <statements>
    GO TO s#4
C
s#3  CONTINUE
    <statements>
C
s#4  CONTINUE
```

(***) Format the ELSE statement as follows:

```
IF ( <expression> ) THEN
  statements
ELSE
  statements
END IF
```

(***) Format the ELSE IF statement as follows:

```
IF ( <expression> ) THEN
  statements
ELSE IF ( <expression> ) THEN
  statements
ENDIF
```

(***) Format the computed GO TO statement as follows:

```
      GO TO ( s#1, s#2, s#3 ) i
C
s#1  CONTINUE
      <statements> )
      GO TO s#4
C
s#2  CONTINUE
      <statements> )
      GO TO s#4
C
s#3  CONTINUE
      <statements> )
C
s#4  CONTINUE
```

Guidelines:

(**) Use the ELSE IF statement in lieu of an Arithmetic IF statement.

(**) Use the ELSE IF statement in lieu of the Computed GO TO statement.

4.12.5 SAVE Statement

Standards:

(***) To retain the value of a variable in a program unit after control is returned from that unit, use a SAVE attribute or SAVE statement. Do not rely on the compiler to retain the variable's value for you. By explicitly putting the variable in a SAVE statement, it will help make it clear to anyone reading the code that the value of the variable is being retained between calls to the routine.

4.12.6 EQUIVALENCE Statement

Guidelines:

(**) The use of EQUIVALENCE statements is discouraged. The TRANSFER intrinsic function should be used instead of EQUIVALENCE statements.

4.12.7 END Statement

Standards:

(***) An END statement shall be used as the last statement for all functions, subroutines, modules and programs.

(***) The last statement in a DO loop shall be an END DO statement. If there are a large number of statements in a DO loop, include a comment that relates the END statement to the DO statement for that loop.

(***) The last statement in an IF loop shall be an END IF statement. If there are a large number of statements in an IF loop, include a comment that relates the END statement to the IF statement for that loop.

4.13 Common Libraries

Recommendations:

(*) Use the IMSL Fortran Numerical Library. This library is usually available from suppliers of Fortran compilers. The IMSL Fortran Library, a complete collection of mathematical and statistical algorithms for high performance computing applications, integrates the IMSL F90 Library with the IMSL Fortran 77 library into a single, cohesive package. The IMSL Fortran Library includes all of the algorithms from the IMSL Family of Fortran libraries for more than three decades, including the IMSL F90 Library, the IMSL Fortran 77 Library, and the IMSL parallel processing features.

4.15 Memory Allocation

Guidelines:

(**) Use dynamic allocation of memory wherever possible.

(**) Do not allocate memory for local variables until they are used in a subprogram, and deallocate the memory for a local variable as soon as its use in the program is finished.

4.16 Documentation

Standards:

(***) Accompany each program unit END statement with a comment indicating the program unit name.

Guidelines:

(**) Delimit comments consistently, with a 'C' in column one (1) and a blank line before the comment line.

(**) Precede each conditional statement (e.g., DO, IF, Computed GO TO) with a block comment describing the condition being tested and the branching alternatives.