

CIMSS FORTRAN-77 Guidelines

General Guidelines

1. High level languages are a convenience for people, not computers, so write your programs as if people mattered.
2. Give variables names that humans can understand. It only inconveniences humans when a variable is named 'TXX' instead of 'temperature'; the compiler doesn't care.
3. Except for loop indices, variables should not be re-used within a routine.
4. Do not assume that variables in subroutines or functions retain their values between calls. Use SAVE statements for variables that must retain their values between calls, and assume that all other variables are undefined until initialized.
5. Only SAVED variables are guaranteed to retain their value between module calls.
6. Do not use compiler options to perform functions that can be handled by the language. For example, use SAVE statements to statically allocate and save local variables rather than a compiler option such as -static.
7. Use in-line formatting instead of numbered FORMAT statements.
8. Double precision and real variables should not be compared for strict equality with other double precision or real variables (i.e., using .EQ., .NE.).

Declarations

1. IMPLICIT NONE statement should appear in each routine.
2. All variables should be explicitly declared.
3. All variables should be initialized prior to use.
4. Use generic type statements like REAL, INTEGER, and DOUBLE PRECISION instead of non-standard statements like REAL*4, INTEGER*4, REAL*8.
5. PARAMETER variables should not be redefined.
6. If COMMON must be used, separate them into include files which are inserted into the code using INCLUDE. All COMMON blocks should be named.
7. EQUIVALENCE statements should not be used.
8. Declarations should be ordered consistently throughout the code. An example of consistent ordering is:
 - a) Module arguments in the same order as the argument list,
 - b) Global variables in COMMON (using INCLUDE files),
 - c) Local PARAMETERS (types and values),
 - d) Local variable types,
 - e) User defined functions,
 - f) EXTERNAL declarations,
 - g) DATA statements.

Structure and Style

1. There should be sufficient white space to make the code readable. Blank lines between each major code structure are encouraged.
2. Use a consistent comment style to highlight code structure and increase readability.
3. Use mixed case rather than all upper case.
4. Computed and arithmetic GOTOs should not be used.
5. Loop control variables should be of INTEGER type.
6. Unconditional branching (GOTO) should only be used within nested structures.

7. Computed and arithmetic GOTOs should not be used.
8. DO-loops should be terminated with CONTINUE or ENDDO. The index of a DO loop should always be of integer type, and the index should not be modified inside the loop.
9. Generic intrinsic functions should be used rather than type specific functions.
10. The correct functioning of code should not depend on the rounding behavior of converting a DOUBLE PRECISION or COMPLEX to other floating types.
11. If labels must be used, use a consistent labeling scheme with labels increasing in value through a module.

Language Extensions

Source code should comply with the ANSI standard specification for FORTRAN 77 or FORTRAN 90. The following extensions to FORTRAN 77 may be used:

1. INCLUDE statement,
2. BYTE and INTEGER*2 data types,
3. DO WHILE,
4. ENDDO,
5. STRUCTURE data types,
6. Names up to 31 characters in length,
7. IMPLICIT NONE statement,
8. Block IF with ELSE IF and END IF,
9. Extended character set to include lower case letters, underscore ("_"), left and right angle bracket ("[" and "]"), quotation mark ("\"), exclamation point ("!"), percent sign ("%"), and ampersand "&"),
10. Long line extensions beyond 72 characters per line,
11. Bit manipulation including: Inclusive OR, Logical AND, Logical Complement, Exclusive OR, Logical Shift, Circular Shift, Bit Extraction, Bit Move, Bit Testing, Set Bit, Clear Bit, and Bit Constants.

Obsolete Language Features

Constructs and features of the Fortran 90 language that are marked for removal in the next release of the FORTRAN standard should **NOT** be used:

1. Arithmetic IF,
2. Real and double precision DO control variables and DO loop control expressions,
3. Shared DO loop termination,
4. DO loop termination on a statement other than END DO or CONTINUE,
5. Branching to an END IF statement from outside its IF block,
6. Alternate Return,
7. PAUSE statement,
8. ASSIGN and assigned GO TO statements,
9. Assigned FORMAT specifiers,
10. cH edit descriptor.

Compiler Options

The compiler should always be used to the fullest extent to detect coding errors.

Recommended options include:

1. Generate debugging information (allows the debugger to be used),
2. Optimization off (make sure you get the right answers before you optimize),

3. No implicit data typing,
4. Check array bounds at run time,
5. Enable all compiler warnings,
6. Enable floating point traps (e.g. for divide by zero; usually disabled by default),
7. Allow source lines longer than 72 characters.

The appropriate compiler switches are:

SunOS: `-g -u -C -e -fsimple=0 -Xlistv4 -XlistE \ -ftrap=%all,no%inexact`

IRIX: `-g -u -C -coll20 -n32 -fullwarn -bytereclen -lfpe`

AIX: `-g -u -C -qfixed=120 -qrndsngl -qnomaf -qhalt=W \ -qflttrap=overflow:underflow:enable \ -qflttrap=zerodivide:invalid:enable`

To activate floating point traps in IRIX, set the environment variable TRAP_FPE to "ALL=ABORT".

Code Checkers

Once your code compiles cleanly, run `ftnckek` to detect semantic errors in a Fortran program that a compiler usually does not. `ftnckek` is available from <http://www.dsm.fordham.edu/~ftnckek>

Recommended `ftnckek` switches are as follows:

```
'-declare -nodivision -noextern -nolibary '\
'-nolist -portability -nopproject -nosymtab -noverbose '\
'-f77=all -f77=no-do-endo -f77=no-include'\
'-f77=no-long-name'
```

Code Polishers

The NAG FORTRAN polishing and restructuring tools are available on origin.ssec.wisc.edu. See the man pages of these routines for details:

Analysers		Transformers	
<code>nag_pfort</code>	Portability Verifier	<code>nag_apt</code>	Precision Transformer
<code>nag_fcalls</code>	Call tree lister	<code>nag_chname</code>	Name Changer
<code>nag_fxref</code>	Cross referencer	<code>nag_dec</code>	Declaration Standardiser
<code>nag_libdoc</code>	Library documentor	<code>nag_lvi</code>	Local Variable Initialiser,
		<code>nag_polish</code>	Pretty Printer,
		<code>nag_profile</code>	Profiler,
		<code>nag_struct</code>	Restructurer.

Debugging with dbx

Instead of the traditional method of inserting 'print' statements in your code to guess where your program fails, try using `dbx`, e.g.

```
$ dbx myprog.exe
dbx version 7.2 Aug 29 1997 03:27:55
Executable /modishome/gumley/src/myprog.exe
(dbx) stop in exit
Process 0: [3] stop in exit
(dbx) run
Process 6443 (myprog.exe) started
```

When your program stops (say with a divide by zero), type

```
(dbx) up
(dbx) up
```

until the relevant FORTRAN or C source line appears. Then you can examine the values of variables, e.g.

```
(dbx) print i, xdata
```

Other useful `dbx` commands are

```
(dbx) stop in proc      (stop at the first line of routine proc)
(dbx) stop var          (stop when variable var changes)
(dbx) stop at n         (stop at source line number n in current routine)
(dbx) step n            (execute the next n source lines)
(dbx) cont              (continue execution)
(dbx) status            (print current breakpoints)
(dbx) trace proc        (when proc is called, print caller and arguments)
```

Numerical Libraries

For common mathematical operations (e.g. matrix inverse) use well-tested freely-available numerical libraries, no matter how much fun it might be to write your own:

BLAS: High quality "building block" routines for basic vector and matrix operations. <http://www.netlib.org/blas/index.html>

FFTPACK: Fast Fourier transform of periodic and other symmetric sequences. It includes complex, real, sine, cosine, and quarter-wave transforms. <http://www.netlib.org/fftpack/index.html>

LAPACK: Routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, singular value problems, and associated matrix factorizations (LU, Cholesky, QR, SVD, Schur). <http://www.netlib.org/lapack/index.html>

SLATEC: Over 1400 general purpose mathematical and statistical routines. <http://www.netlib.org/slatec/index.html>

CMLIB: Over 750 routines for problems in many areas of mathematics and statistics. <http://math.nist.gov/mcsd/Software.html>