

Safety and Portability of IERS Conventions Software

Michael Gerstl, DGFI

Fortran 77

“Fortran 77” = extension to the ANSI FORTRAN 77 standard.

Responsibility :

American National Standards Institute (ANSI X3J3)

International Organization for Standardization (ISO/IEC JTC1 SC22 WG5)

Fortran 77 (extended) is without standardization and with many dialects.

Obsolescent language elements :

- compromising safety and portability : implicit variable declaration, double precision, ...
- disabling the compiler optimisation : equivalence, noninteger do loop variables, multiple loops terminating in a single statement, jumps from outside into loops, ...
- hindering the parallelization : data statement, ...

State of the Conventions software :

```
SUBROUTINE ARG(IYEAR, DAY, ANGLE)
  IMPLICIT DOUBLE PRECISION (A-H, O-Z)
  REAL ANGFAC(4, 11)
  DIMENSION ANGLE(11), SPEED(11)
C
C  SPEED OF ALL TERMS IN RADIANS PER SEC
C
  EQUIVALENCE (SPEED(1), SIGM2), (SPEED(2), SIGS2), (SPEED(3), SIGN2)
  EQUIVALENCE (SPEED(4), SIGK2), (SPEED(5), SIGK1), (SPEED(6), SIGO1)
  EQUIVALENCE (SPEED(7), SIGP1), (SPEED(8), SIGQ1), (SPEED(9), SIGMF)
  EQUIVALENCE (SPEED(10), SIGMM), (SPEED(11), SIGSSA)
  DATA SIGM2/1.40519D-4/
  DATA SIGS2/1.45444D-4/
  DATA SIGN2/1.37880D-4/
  ...
```

State of the Conventions software

Correct only SOFA software (implicit none!)

Programming errors become manifest when a program is transferred

- 32-bit processor → 64-bit processor,
- Fortran 77 → Fortran 90.

The double precision problem

	32-bit processor standard default real = 4 byte	64-bit processor compiler-dependent default real	
		= 4 byte	= 8 byte
single precision	4 byte	4 byte	8 byte
double precision	8 byte	8 byte	16 byte

The 4-byte number model according to the IEEE standard 754: A floating point number is represented with a significand or mantissa occupying 24 bits.

Relative error of size

$$2^{-24} \approx 5.96 \times 10^{-8}.$$

This quantity is known as **unit roundoff**.

By error propagation and above all accumulation this roundoff error can increase by one to two orders of magnitude. Thus, the final relative error can distort the 7th decimal place.

Prevalent error cause: Variables have been forgotten to be declared, because `implicit none` is missing, literal constants with forgotten type.

```
...
S = 218.31664563D0 + 481267.88194D0*T - 0.0014663889D0*T**2
1  + 0.00000185139D0*T**3
PR = 1.396971278*T + 0.000308889*T**2 + 0.000000021*T**3
1  + 0.000000007*T**4
S = S + PR
H = 280.46645D0 + 36000.7697489D0*T + 0.00030322222D0*T**2
1  + 0.000000020*T**3 - 0.00000000654*T**4
P = 83.35324312D0 + 4069.01363525D0*T - 0.01032172222D0*T**2
...
```

Example from the IERS Conventions:

The first literal constant in the 3rd line will be stored as a single precision constant of type default real.

$$\text{PR} = 1.396971278000000 \dots$$

 ↑ ↑
 4-byte 8-byte roundoff

If default real is set to 4 bytes, the roundoff will cut off significant digits.

Not a safe remedy is

- compiler option `-r8`
- machine-independent type `real*8`
- compiler option like `-default-real=8`

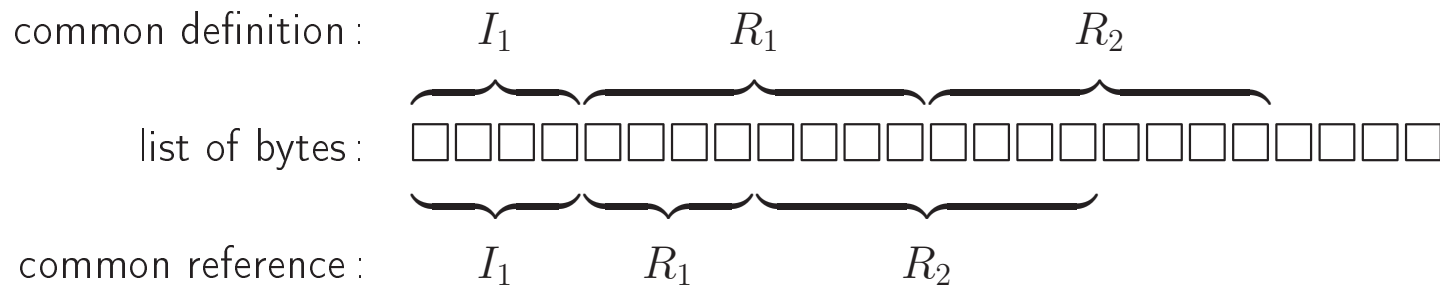
The double precision problem has been finally solved not until Fortran 90.

Unsafe common blocks

The language does not distinguish between **definition** and **reference** of a common block.
Each reference is a new definition.

Mapping between definitions is established byte-wise.

Due to byte-wise mapping, two definitions of the same common may differ intentionally or accidentally.



Common blocks should be substituted.

There is no solution within Fortran 77.

Portable Fortran 77 programs

Fortran 77 program which runs on different machines and under different Fortran dialects.

```
...
#ifdef INTEGER_8
    KBIT = 64
    KNEG = -9223372036854775807
#elif (defined VAX)
    KBIT = 32
    KNEG = -2147483648
#else
    KBIT = 32
    KNEG = -2147483647
#endif
...

DO 220 JN = JM , KTRUNC
    INDEX = INDEX + 2
    IF ( JN .GE. KSUBSP ) THEN
#ifdef (VAX) || defined (IBM) || defined (__alpha) || defined hpR64 ...
        ZNORM(JN) = MAX(ZNORM(JN), DBLE(ABS(PSPEC(INDEX))))
        ZNORM(JN) = MAX(ZNORM(JN), DBLE(ABS(PSPEC(INDEX+1))))
#else
        ZNORM(JN) = MAX(ZNORM(JN), ABS(PSPEC(INDEX)))
        ZNORM(JN) = MAX(ZNORM(JN), ABS(PSPEC(INDEX+1)))
#endif
    ENDIF
220 CONTINUE
```

Preprocessor commands defining and using logical switches and including common machine-adapted code.

Disappearance of Fortran 77 compilers

GNU Fortran compiler **gfortran** alias **g77**

Two lines of development:

1. **g77**, a Fortran 77 compiler extended to understand Fortran 95 elements.

Never be fully Fortran 95 compatible.

Example: $F = 1.0$ for an array F of dimension $(1:N)$.

Fortran 77: $F(1) = 1.0$ customarily,

Fortran 90: $F(1:N) = 1.0$ pursuant to standard.

2. **g95**, a Fortran 95 compiler extended downwards to Fortran 77.

Not all programs that could be compiled with **g77** pass **g95**.

The compiler can't properly work on programming tricks not intended for Fortran 77.

In addition, there is no compiler option `-std=f77` for a non-standardized language.

In the past : `gfortran = g77`.

Now (since gcc 4.x): `gfortran = g95`, support for **g77** stopped.

I think it's high time to migrate to a new programming language.

The new Fortran

1987–1991 the ISO WG5 and ANSI X3J3 fight for a safe, machine-independent, and compiler-independent programming language. The first outcome in 1991 was Fortran 90.

Fortran 90 is a completely new high-level problem oriented language.

Three revisions: Fortran 90, Fortran 95, and Fortran 2003.

Binary compatibility within Fortran code is possible with Fortran 95, if IEEE standard arithmetic is used.

Binary compatibility to C was reached with Fortran 2003.

For IERS Conventions the Fortran 95 standard is sufficient.

Migration

Fortran 90 can be used as

- procedural language (as the former Fortran), or as
- object-oriented language (since Fortran 2003), or intermediate, as
- module-oriented language.

A **module** is a program unit which contains data and procedures (“methods”) which handle those data.

- Compared with object-oriented programming the modules are single instances of classes which are build at compile-time.
- They are safer and run faster as objects.
- The module is the safe and portable successor of the **common block**.

Since most of you use procedures and common blocks (which are outmoded), the modul-oriented approach provides the most easy way of migration.

My personal migration schedule looks like as follows.

1. Rewrite the declarative header of each routine using `implicit none`, machine-independent types, and attributes.
2. Compile and run. You have to resolve lots of errors that are reported by the compiler.
3. Tag all real and complex literal constants with a type.
For example: `360.0` \longrightarrow `360.0_r8`.
4. Convert `common` into `module` and replace, if possible, data statements by data and parameter attributes.
(Fortran 90 is capable of parameter arrays!)
5. Replace obsolescent features.
6. Vectorization.
7. Transformation of appropriate routines into module subprograms, internal subprograms, or intrinsic subprograms.
8.

Benefits of migration

- The code is still legible for someone who is only familiar with Fortran 77.
- The code is shorter: The definition of a variable, which was formerly dispersed over several lines, is now concentrated in a single line:

```
Fortran 77: integer IFL  
           dimension IFL(10)  
           data (IFL(i),i=1,10) / 0,1,1,1,2,2,2,2,2,2 /
```

```
Fortran 90: integer :: IFL(1:10) = (/ 0,1,1,1,2,2,2,2,2,2 /)
```

```
Fortran 03: integer :: IFL(1:10) = [ 0,1,1,1,2,2,2,2,2,2 ]
```

- Vectorial notation increases again the legibility of code.

```
Fortran 77: do i = 1,N  
           F(i) = F(i) + dF(i)  
           enddo
```

```
Fortran 90: F = F + dF  
           or: F(1:N) = F(1:N) + dF(1:N)
```

- Lots of errors, which formerly appeared occasionally at run time, are detected during compilation. This is the consequence of strict language definition.
- Altogether, that implies less maintenance effort when a program has to be modified.
- In addition, you can realize an astonishing gain in velocity. Through the conversion in 1994, our satellite orbit perturbation program decreased in runtime by 50 per cent.

Warning

Results from Fortran 77 and Fortran 90 are not comparable to the last digit. Fortran 90 has intrinsic vector and matrix operations, the accuracy of which can never be reached by Fortran 77.

Remarks

I provide assistance with migration of conventional software.

One of my colleagues persistently refuses to convert his program because he must integrate conventional software as it is, in order to be compatible. Here we move in a circle.

Literature

Unfortunately I do not know a recommendable book to learn Fortran 90. As a reference book, I use

Wilhelm Gehrke: Fortran 95 Language Guide. Springer, London 1996