

# **The Current Status of Fortran 90 and Fortran 95**

*John Reid*

*CISD, Rutherford Appleton Laboratory*

**16 January 1996**

## Abstract

Fortran 90 replaced Fortran 77 (F77) as the International (ISO) and American (ANSI) Fortran standard in 1991. It is an upward-compatible extension of Fortran 77, which will preserve the vast investment in existing codes. Of particular benefit for scientific computing are dynamic storage, assumed-shape arrays, optional arguments, structures, and pointers. Software can be better engineered with interfaces that are much more user friendly.

I tell you something of my personal experience in using compilers (I have access to the Nag, EPC, Fujitsu, DEC, IBM, and CRAY compilers), and explain the design of two packages that exploit the new features and are part of the latest release of the Harwell Subroutine Library. One is for automatic differentiation and the other for frontal solution of finite-element equations.

Fortran 95 is a minor extension of Fortran 90. Its design was finalized in November 1995 and it is now in its final stages of acceptance as a standard. I will explain the differences between Fortran 95 and Fortran 90.

## History

- 1966 First computer language standard
- 1978 Fortran 77 standard
- 1987 First draft *400 letters*
- 1988 ISO Paris meeting breaks deadlock
- 1989 Second draft *150 letters*
- 1990 Third draft *29 letters*
- 1991 ISO standard
- 1995 Fortran 95 design finalized
  
- 1996 (Oct) Fortran 95 ISO standard
- 2000 Fortran 2000 design finalized
- 2001 Fortran 2000 ISO standard

## **Present situation re compilers**

Nag: Versions for most Unix platforms, VMS, PCs (with Salford).

IBM: For RS/6000. HPF coming.

EPC: Optimizing compiler for SUN, RS/6000, MIPS, ICL DRS6000, SGI.

Cray: For Crays and SUNs

Sun: From Cray.

DEC: Optimizing compiler, with HPF.

Fujitsu: For SUNs.

Lahey: For PCs.

SGI: For SGI.

HP: with EPC for HP and Convex.

NaSoftware: For PCs and SPARC, with HPF.

Microsoft: Uses Nag front end.

+ More

## Books

ISO, *Fortran 90 (standard)*, ISO Publications Dept.,  
Case Postale 56, 1211 Geneva 20, or  
walt@atnetcom.com.

Metcalf and Reid, *Fortran 90 explained*, OUP (also  
available in French, Japanese and Russian).

Counihan, *Fortran 90*, Pitman.

Adams, Brainerd, Martin, Smith, and Wagener,  
*Fortran 90 handbook*, McGraw Hill.

Brainerd, Goldberg, and Adams, *Programmer's guide  
to Fortran 90, second edition*, Unicomp.

Morgan and Schonfelder, *Programming in Fortran 90*,  
Alfred Waller.

Kerrigan, *Migrating to Fortran 90*, O'Reilly  
Associates.

Wille, *Advanced Scientific Fortran 90*, Wiley.

Meissner, *Fortran 90*, PWS Kent, Boston.

Schick and Silverman, *Fortran 90 and Engineering  
Computation*, Wiley.

Chamberland, *Fortran 90, A Reference Guide*,  
Prentice Hall.

Hahn, *Fortran 90 for Scientists and Engineers*,  
Edward Arnold.

Gehrke, *Fortran 90 Language Guide*, Springer.

Ellis, Philips, Lahey, *Fortran 90 Programming*,  
Addison Wesley.

Adams, Brainerd, Martin and Smith, *Fortran Top  
90-Ninety Key Features of Fortran 90*, Unicomp.

Chivers and Sleightholme, *Introducing Fortran 90*,  
Springer-Verlag.

Vowels, *Introduction to Fortran 90, Algorithms, and  
Structured Programming, Vol. 1* Vowels.

Ortega, *Introduction to Fortran 90 for Scientific  
Computing*, Saunders College Publishing.

Smith, *Programming in Fortran 90*, Wiley.

Mayo and Cwiakala, *Schaum's Outline of Theory and  
Praxis – Programming in Fortran 90*, McGraw Hill.

Redwine, *Upgrading to Fortran 90*, Springer-Verlag.

More in other languages.

## **comp-fortran-90**

Bernd Eggen, formerly of University of Sussex, has established an email platform for discussion of Fortran 90 and HPF.

To join, send a message to

mailbase@mailbase.ac.uk

containing as its only line:

join comp-fortran-90 *joe bloggs*

## **World Wide Web**

For information on Fortran 90 products:

<http://www.fortran.com/fortran>

## **Libraries**

Nag: A collection of modules, based on their Fortran 77 library

Visual Numerics (IMSL): A collection of modules, emphasizing use of defined operators

## Tools

Nag: A collection of simple tools, including a verifier and a source-form translator.

PSR: VAST-90 – converts to and from Fortran 77.

[info@psrv.com](mailto:info@psrv.com)

APR: FORGE 90.

[forge@netcom.com](mailto:forge@netcom.com)

Parasoft: Converts to Fortran 77.

[f90-info@parasoft.com](mailto:f90-info@parasoft.com)

Metcalf: Fortran 90 program that converts Fortran 77 program to new source form, tidies, makes interfaces.

<ftp://jkr.cc.rl.ac.uk/pub/MandR>

## **Plan for this talk**

- (a) Summary of Fortran 90
- (b) Ex 1. Automatic differentiation
- (c) Ex 2. Frontal solutions
- (d) Experience with compilers
- (e) Fortran 95

## Design objectives

- greater expressive power – *arrays*.
- enhanced safety – *implicit none*.
- enhanced regularity – *real/double*.
- extra fundamental features – *dynamic storage*.
- fix Fortran 77 problems – *nonadvancing i/o*.
- better hardware exploitation – *arrays*.
- improve portability – *parameterized types*.

## Language evolution

- **No deleted features**

Moving to a f90 compiler is like moving to a f77 compiler with extensions.

- **The following are obsolescent and might be deleted from the next standard:**

- Arithmetic IF
- Noninteger DO index \*
- DO termination other than CONTINUE or END DO
- Shared DO termination
- Alternate return
- PAUSE \*
- ASSIGN and assigned GO TO \*
- Assigned FORMAT specifiers
- Branching to END IF \*
- H edit descriptor \*

\* Planned for deletion in Fortran 95

## Source form

A = B; C = C + D ! This is a comment

Case insignificant except in character context

Names up to 31 chars, including \_

" and ' for character delimiters

Character set includes: ! " % & ; ? [ ]

### New form only:

Cols 1, 6, 72 of no significance

& for continuation:

a = b &

+ c

No blanks in tokens, except END IF, etc.

Lines up to 132 chars

Statements up to 40 lines

## Dynamic storage

### (a) Automatic

```
SUBROUTINE F90(N)
  INTEGER N
  REAL W(N)
```

### (b) Allocatable

```
REAL, ALLOCATABLE :: A(:)
  READ(*,*) N
  ALLOCATE (A(N))
  . . .
  DEALLOCATE (A)
```

### (c) Pointers

```
REAL, POINTER :: A(:)
```

## Array features

- Whole array expressions and assignments
- Elemental intrinsics
- Transformational intrinsics
- Array constants
- Array-valued functions
- Zero-sized arrays
- Array constructors
- Array sections
- WHERE
- Assumed-shape dummy arguments

## Example

$$\sum_{i=1,n} a_i \cos(x_i)$$

SUM ( A \* COS(X) )

Notes:

- (i) SUM is transformational
- (ii) \* and COS are elemental

## Procedures

- May be recursive
- May be internal (one level)
- Keyword calls
- Optional arguments
- Intent
- Generic name
- Assumed-shape array arguments
- Interface blocks for external and dummy procedures

## Structures and derived types

- Structures allow all the data pertinent to a particular problem to be collected in one object.
- Derived types allow existing code (with +, −, /, \*, ...) to be reused for
  - intervals
  - values and their derivatives (automatic differentiation)
  - strings
  - etc.

## Type parameters

- All intrinsic types have a ‘kind’ parameter
- Precision can be parameterized
- Intrinsic to enquire about the environment
- Intrinsic to manipulate reals
- Complex no longer second class

## Pointers

- Automatic dereferencing
- Notation for pointer assignment:  
    `POINTER => TARGET`
- Targets must be other pointers or have the target attribute
- Type and rank agreement
- Pointer dummy arguments and function results
- Allocate and deallocate

## Modules

Collections of

- data
- types
- procedures
- procedure interfaces

Definitions made once and then used:

USE JKR

Libraries likely to become libraries of modules.

Advantages:

- name hiding
- private data
- interface checks
- single definition
- generic names

Note: modules must be visible at compile time.

## Automatic differentiation

Joint work with John Pryce and David Cowey of RMCS.

If  $b$  and  $c$  are variables that depend on the independent variables  $x_i, i = 1, 2, \dots, n$ , and their derivatives are known, we can calculate the derivatives of

$$a = b * c$$

by the chain rule,

$$\frac{\partial a}{\partial x_i} = \frac{\partial b}{\partial x_i} * c + b * \frac{\partial c}{\partial x_i}$$

Define a derived type that holds values and derivatives. Overload all the operators with functions that apply the chain rule.

## How to use HSL\_AD01

- (i) add USE statement
- (ii) declare all independent and dependent variables to be of type AD01\_REAL with initial value AD01\_UNDEFINED
- (iii) initialize the module, specifying the independent variables  $\mathbf{x}$  and their values
- (iv) compute values of the dependent variables
- (v) subroutine calls yield the required derivatives

## Language supported

- (i)  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $**$ ,  $==$ ,  $/=$ ,  $>=$ ,  $>$ ,  $<=$ ,  $<$   
 $ad \bullet ad$ ,  $ad \bullet r$ ,  $ad \bullet i$ ,  $r \bullet ad$ ,  $i \bullet ad$
- (ii)  $\text{atan2}$ ,  $\text{max}$   $\text{min}$ ,  $\text{sign}$   
 $ad \bullet ad$ ,  $ad \bullet r$ ,  $r \bullet ad$
- (iii)  $-$ ,  $\text{abs}$ ,  $\text{acos}$ ,  $\text{asin}$ ,  $\text{atan}$ ,  $\text{cos}$ ,  $\text{cosh}$ ,  $\text{exp}$ ,  $\text{log}$ ,  
 $\text{log10}$ ,  $\text{sin}$ ,  $\text{sinh}$ ,  $\text{sqrt}$ ,  $\text{tan}$ ,  $\text{tanh}$   
 $ad$
- (iv)  $\text{aint}$ ,  $\text{anint}$ :  $ad \Rightarrow r$
- (v)  $\text{int}$ ,  $\text{nint}$ :  $ad \Rightarrow i$
- (vi)  $=$ :  $ad = ad$ ,  $ad = r$ ,  $ad = i$ ,  $i = ad$   
**NB  $r = ad$  omitted**

No array operations at present

## Fortran 90 version of HSL frontal code MA42

Joint work with Jennifer Scott

**Motivation:** Nested use of MA42, e.g.:

- Subdivide problem in 4,
- use MA42 to produce Schur complements for each on interfaces,
- apply MA42 to the interface problem.

Very complicated because of the need for workarrays, direct-access files, long argument lists.

**Solution: collect everything about a problem  
in one object**

```
USE HSL_MA42
TYPE(MA42_DATA) DATA
CALL MA42_INITIALIZE (DATA)
I = 1
DO
  READ . . .
  CALL MA42_ANALYSE (LIST(1:LEN),DATA)
  IF(FINAL) EXIT
END DO
DO J = 1,I
  READ . . .
  CALL MA42_FACTORIZE (LIST(1:LEN), &
                      REALS(1:LEN,1:LEN),DATA)
END DO
READ . . .
CALL MA42_SOLVE (B,X,DATA)
```

The module looks after everything in DATA,  
except a few control or informative variables.

## Options obtained by:

- Extra calls, e.g.  
`CALL MA42_FILES(LENBUF,LENFILE,DATA)`
- Optional arguments, e.g.  
`CALL MA42_SOLVE(B,X,DATA,TRANS)`
- Setting components of the data structure  
`DATA%ALPHA = 0.001`
- accessing components  
`FLOP_COUNT = DATA%FLOPS`

## **Personal experience with compilers**

Comment: Since I am developing software, I want good diagnostics and good compile speed. Less interested in run speed.

Also, it is not my practice to use interactive debugging.

## 1. EPC – epcf90 on SPARC 10, SunOS

This is my ‘workhorse’.

- fast compilation
- checks unassigned variables, subscripts, interfaces
- optimized object code
- marvellous ‘dumps’ that show the values of all variables accessible at the point of termination and up the call chain
- interactive debugger (Solaris) said to be excellent

## 2. Nag – f90 on SPARC 10, SunOS

- quite fast compilation
- checks subscripts, interfaces
- execution speed improving
- user is unaware of C intermediate code, but it is sometimes helpful to look at it.

### **3. Fujitsu – frt on SPARC 10, SunOS**

- most robust of them all
- marvellous checks on unassigned variables, subscripts, interfaces
- pretty printing
- checks on which statements executed
- poor compilation speed
- wide range of optimization choices
- very slow execution with full checks

#### **4. CRAY – f90 on Y-MP**

- robust
- good optimization of Fortran 77 code
- using modules awkward

#### **5. IBM – xlf on RS/6000**

- robust
- good optimization

#### **6. Digital – f90 on unix alpha**

- profiling feature very useful
- wide range of optimization choices

## Plans for Fortran 95

Minor revision that incorporates items in the corrigenda and editorial improvements, and these new features:

- FORALL
- Nested WHERE
- Pure procedures
- Elemental procedures
- Pointer initialization
- Automatic component initialization
- Allocation status always defined
- User procedures in specifications
- CPU\_TIME
- Minor changes

## **New obsolescent features**

- Computed GO TO
- Statement functions
- DATA among executables
- Assumed-length character functions
- Fixed source form
- Character \* declarations

## Forall

It is essentially an array assignment expressed with the help of indices. Some examples:

$$\text{FORALL (I=1:N) A(I, I) = X(I)}$$
$$\text{FORALL (I=1:N, J=1:M) A(I, J) = I+J}$$
$$\text{FORALL(I=1:N, J=1:N, Y(I, J)/=0.) \&}$$
$$\text{X(J, I) = 1.0/Y(I, J)}$$

The FORALL construct also exists.

## Pure procedures

Free of side effects:

- (i) if a function, does not alter any dummy argument;
- (ii) does not alter any part of a variable accessed by host or use association;
- (iii) contains no local variable with the SAVE attribute;
- (iv) performs no operation on an external file;
- (v) contains no STOP statement.

Needed for procedures called in a FORALL.

Very strict rules to ensure that purity can be checked at compile time.

Suitable for specification expressions.

## **Elemental procedures**

Written for scalars, but may be called for arrays.  
Must satisfy the requirements for a pure procedure.

In addition, all dummy arguments and function results must be scalar variables without the pointer attribute.

## Pointer initialization

```
REAL, POINTER :: VECTOR(:) => NULL()
```

## Automatic object initialization

Values declared in the type specification, e.g.,

```
TYPE ENTRY
```

```
  REAL :: VALUE = 2.0
```

```
  INTEGER INDEX
```

```
  TYPE(ENTRY), POINTER :: NEXT=>NULL()
```

```
END TYPE ENTRY
```

Dearly needed for HSL\_AD01.

## **Allocation status always defined**

The undefined status, which cannot even be tested for, is abolished, though the status is sometimes processor defined.

## Technical reports

Technical reports are being developed for

- Handling floating point exceptions
- Calls to C
- Allocatable dummy arguments, function results, and structure components

It is intended that these be completed in 1996 and for the features to be adopted as part of Fortran 2000 unless snags are found.

A 'beta test' for new language features.

## Fortran 2000

Plans are still fluid. Likely to include:

- Exception handling
- Enhancements to allocatable arrays
- Calling C
- Parameterized data types
- Derived type I/O
- Minor enhancements

Will probably be a small revision (IMHO)

## Conclusions

Fortran 90 is here:

- compilers widely available
- plenty of books
- tools appearing
- evolving (Fortran 95, Fortran 2000)