NAGWare f95 and reliable, portable programming.

Malcolm Cohen

The Numerical Algorithms Group Ltd., Oxford

"How to detect errors using NAGWare f95, and how to write portable, reliable programs. Support for features from the latest Fortran standard and plans for future releases."

Contents

1. The Fortran Standard:
   - portable programming;
   - modern programming;
   - language development.

2. NAGWare f95 and the Fortran Builder:
   - detecting errors at compile time;
   - detecting errors at run time;
   - Fortran 2003 features;
   - future plans.

Portable programming

A portable program

- can be run on many systems without source code changes;
- gives correct results on those systems.

Portable programming reduces the lifetime cost of the program:

- no need to have different versions on different machines;
- reduces amount of maintenance;
- increases confidence in correctness.

The Fortran Standard

- features that must be supported by all Fortran compilers;
- precise definitions so that the features work the same way on all compilers.

All Fortran compilers have extra features; but using an extra feature means that if you try to use the program on another machine

- the other compiler might not have the feature;
- even if it has the feature, it might not work the same way.

History

1954 Fortran project starts at IBM.

1966 Fortran 66, the first programming language standard.

1978 Fortran 77; modernisation begins.

1991 Fortran 90 (major revision). The basis of modern Fortran.

1997 Fortran 95 (minor revision).

2004 Fortran 2003 (major revision). Object-oriented and more.

Modern Programming

Modern Fortran is:

- easier to write,
- more reliable (newer features are less error-prone),
- more powerful,
- efficient,
- supported by all the major manufacturers.

Modern Fortran Key Features (1)

Usability

- Long names (6→31 characters).
- Free format source form.
- Modern control structures.
- Modern data structures.
- Modules.

Modern Fortran Key Features (2)

Functionality

- Dynamic memory allocation; especially **allocatable arrays**.
- Array expressions and assignment.
- Powerful intrinsic functions.

Modern Control Structures

- Generalised DO loop
  (including DO WHILE, EXIT, CYCLE);

- SELECT CASE construct.

Reduces the need for GOTO, and therefore

- makes code easier to read;

- reduces errors.

Modern Data Structures

- Derived types are structures.

- Components can be arrays or scalars.
  They can be of intrinsic types (e.g. Real) or derived types.

- Components can be pointers.

```
Type line
  Integer              :: start(2), end(2)
  Real                 :: width
  Type(Colour)         :: colour
  Type(line),Pointer   :: next_line
End Type
```

Modules

- A module can contain named constants, type definitions, variables, and procedures.

- Defined once and then used anywhere; avoids possible mistakes with multiple definitions.

- Calls to module procedures can be checked at compile time.

- Accessed with a USE statement.

A Simple Module

```
Module int64_module
  Integer,Parameter :: int64 = selected_int_kind(18)
Contains
  Integer(int64) Function gcd(a,b) ! Greatest Common Divisor
    Integer(int64),Intent(In) :: a,b
    ...
  End function
End Module

Program Example
  Use int64_module
  Integer(int64) x,y
  ...
  Print *,gcd(x,y)
End
```

Allocatable Arrays

- Dynamic allocation.
- No need for pointers – fast.
- Automatic deallocation – safe.
- STAT= option to handle failure.

Allocatable Array Example

```
Real,Allocatable :: workspace(:)
...
Allocate(workspace(n*4+10),Stat=istatus)
If (istatus==0) Then
  Call Solve_problem(...,workspace)
Else
  Print *,'Cannot allocate workspace, error code',istatus
End if
```

Language Development

- The Fortran Standard is frequently revised.
- Revisions always aim for backwards compatibility.
- Vendors develop via the standard to reduce risk.
- Key features of Fortran 2003:
  - Allocatable components.
  - IEEE arithmetic support.
  - Object-oriented programming.
  - Interoperability with C.

Fortran 2003 Design Goals

Overall Goals

1. compatible with Fortran 95;
2. safe and efficient.

Object-oriented Goals

- Simple to describe.
- Simple to use.
- Simple to implement.
- Safe to use: detect errors at compile time, not run time.

NAGWare f95: Overview

- World's first Fortran 90 compiler.
- Fortran 95 + many Fortran 2003 features.
- Fortran Builder development environment (Japan only).
- Detects many errors at compile time.
- Comprehensive checking for non-standard programs.
- Unsurpassed runtime error detection.

Runtime Error Detection

- Normal checking features: array subscripts, null pointers.
- Advanced checking features: procedure calls, dangling pointers, undefined variables.
- Memory allocation tracing.

Procedure call checking - 1

Extra information is passed on a procedure reference:

- type and rank of the expected result,
- number of arguments,
- for each argument,
  - whether it is an expression,
  - class: normal, pointer, allocatable, assumed-shape, value, polymorphic.
  - whether it is a procedure,
  - type, rank,
  - number of elements

Procedure call checking - 2

If there is a mistake in the call to the procedure, the program is terminated with an informative error message.

```
Invalid procedure reference -
Actual argument for dummy argument I is REAL instead of INTEGER
Program terminated by fatal error
In PV, line 1 of file2.f90
Called by S, line 23 of file1.f90
Called by MAIN, line 7 of file1.f90
```

Procedures compiled with `-C=calls` can be mixed with ones compiled without; checking will be done only when both the caller and the called routine are compiled with the option.

Dangling pointers

1. Pointer refers to an unsaved local variable; on return from the procedure, the pointer becomes undefined.

2. Pointer refers to allocated memory; this memory is deallocated without clearing the pointer.

Both of these are quite common in C and C++ programs, and cause mysterious failures and crashes long after the event. These can be very hard to detect without compiler assistance.

Procedures compiled with `-C=dangling` can be mixed with ones compiled without; checking will be done only for pointer assignments in checked routines.

Dangling Pointer Example 1

```
Program Test
  Real,Pointer :: x(:,:)
  Call make_dangle
  x(10,10) = 0
Contains
  Subroutine make_dangle
    Real,Target :: y(100,200)
    x => y
  End Subroutine
End

Reference to dangling pointer X
 - Target was RETURNed from procedure TEST:MAKE_DANGLE
Program terminated by fatal error
In TEST, line 4 of dangle.f90
```

Dangling Pointer Example 2

```
Program dangle2
  Real,Pointer :: x(:),y(:)
  Allocate(x(100))
  y => x
  Deallocate(x)
  y = 3
End

Reference to dangling pointer Y
 - Target was DEALLOCATEd at line 5 of dangle2.f90
Program terminated by fatal error
In DANGLE2, line 6 of dangle2.f90
```

Undefined variables

An **undefined** variable is one

- which has never been given a value, or
- which has lost its value.

Requires the whole program to be compiled with the **-C=undefined** option.

To just detect undefined floating-point variables, the **-nan** option can be used. This is faster, and can be used on parts of a program, but does not print such an informative message.

Undefined Variable Example

```
Program undef1
  Real x(100)
  Read *,n
  Read *,x(1:n)
  Print *,product(x)
End

Reference to undefined variable X
Program terminated by fatal error
In UNDEF1, line 5 of undef1.f90

*** Arithmetic exception:  - aborting
In UNDEF1, line 5 of undef1.f90
```

Memory Allocation Tracing

The -**mtrace** option traces memory allocation and deallocation. With the **f95mcheck** program this can be used to find memory leaks.

```
Program memory_leak
  Real,Pointer :: x(:,:)
  Allocate(x(10,20)) ! Leak
  x = 0
  Allocate(x(3,4))
  Deallocate(x)
  Allocate(x(5,6)) ! Leak
  Allocate(x(20,30))
  x = 3
  Deallocate(x)
End
```

Memory Allocation Tracing

Raw Output

```
[Allocated item 1 (size 1025) = Z'2E0008']
[Allocated item 2 (size 1025) = Z'2E0418']
[Allocated item 3 (size 1025) = Z'2E0828']
[Allocated item 4 (size 800) at line 3 of memleak.f90 = Z'2F0008']
[Allocated item 5 (size 48) at line 5 of memleak.f90 = Z'2F0330']
[Deallocated item 5 (size 48, at Z'2F0330') at line 6 of memleak.f90]
[Allocated item 6 (size 120) at line 7 of memleak.f90 = Z'2F0368']
[Allocated item 7 (size 2400) at line 8 of memleak.f90 = Z'2F03E8']
[Deallocated item 7 (size 2400, at Z'2F03E8') at line 10 of memleak.f90]
[Deallocated item 2 (size 1025, at Z'2E0418')]
[Deallocated item 3 (size 1025, at Z'2E0828')]
[Deallocated item 1 (size 1025, at Z'2E0008')]
```

f95mcheck Output

```
7 allocations
***MEMORY LEAK:
LEAK: Allocation 4 (size 800) = Z'2F0008' at line 3 of memleak.f90
LEAK: Allocation 6 (size 120) = Z'2F0368' at line 7 of memleak.f90
```

Fortran 2003 features: supported now

- Allocatable components.
- IEEE arithmetic support.
- Object-oriented programming.

Allocatable components

- Dynamic sizes for array components.
- More efficient than pointer components.
- Safer than pointer components – automatic deallocation.

```
Type matrix
  Real,Allocatable :: value(:,:)
End type
...
Type(matrix) x
...
Allocate(x%value(100,200))
...
```

IEEE arithmetic support

- IEEE exception handling (e.g. overflow and underflow).
- IEEE operations (e.g. remainder, nextafter)
- IEEE inquiry functions (e.g. `IEEE_IS_NAN`).
- Rounding mode control.
- Halting mode control.

```
Use ieee_arithmetic
...
z = x/y
If (ieee_is_nan(z)) Stop 'Result is Not a Number'
```

Basic Object-Oriented Features

Available now:

- Type extension (single inheritance).
- Polymorphic variables.
- Type selection.

Basic Object-Oriented Summary

- "Type extension" produces a new type by extending an old one. The new type *inherits* the components of the old one.

- A polymorphic variable can have a different (*dynamic*) type at different times. They are always dummy arguments, pointers, or allocatable.

- Type selection detects the dynamic type of a polymorphic variable, and provides direct access to extended components.

NAGWare f95 Future Plans

- Fortran Builder for English Windows.
- More Fortran 2003 features.
- Improved performance.
- Further improvements to error detection.

Fortran 2003 features: next update

- Interoperability with C.
- Stream I/O and other I/O enhancements.
- New intrinsic functions and modules.
- More object-oriented features.
- Many other additions.

We have just started to ship the next update for Linux.

Advanced Object-Oriented Features

Coming soon:

- Cloning.
- Type-bound procedures.
- Generic procedures and operators.

All these are included in the next update.

Conclusion

- The Fortran Standard enables portable programming.
- Using new Fortran features can improve reliability.
- NAGWare f95 has unparalleled error detection.
- NAGWare f95 is in the process of being upgraded to the latest Fortran Standard.

Resources

Slides available on web page:

`http://www.nag-j.co.jp/~malcolm/May2006-J.pdf`

Slides about Modern Fortran programming:

`http://www.nag-j.co.jp/~malcolm/Modern-Fortran-J.pdf`

More slides about Fortran 2003 (in English):

`http://www.nag-j.co.jp/~malcolm/F2003-Illustrated.pdf`

Reference Book (in English): "Fortran 95/2003 Explained"
by Metcalf, Reid and Cohen.