Fortran 90/95 and Computational Physics

Jens Hjörleifur Bárðarson

jensba@raunvis.hi.is

University of Iceland
Overview

- What is Fortran?
- Why Fortran?
- Some Important Things
- Summary
What is Fortran 90?
The Origin

A team lead by John Backus developed Fortran, FORmula TRANslation System, in 1954, one of the earliest high-level languages.
The Origin

A team lead by John Backus developed Fortran, FORmula TRANslation System, in 1954, one of the earliest high-level languages.

1966: The first ever standard for a programming language: Fortran 66

New standard 1978: Fortran 77

The need to modernise the language → Fortran 90/95
Fortran 90

http://csep1.phy.ornl.gov/pl/pl.html
Why Fortran 90?
How does F90 compare?

<table>
<thead>
<tr>
<th>functionality</th>
<th>F77</th>
<th>C</th>
<th>C++</th>
<th>F90</th>
</tr>
</thead>
<tbody>
<tr>
<td>numerical robustness</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>data parallelism</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>data abstraction</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>object oriented programming</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>functional programming</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>average</td>
<td>3.4</td>
<td>3.2</td>
<td>2.2</td>
<td>1.2</td>
</tr>
</tbody>
</table>

http://csep1.phy.ornl.gov/pl/pl.html

One of the ultimate goals of F90 is that the code must be efficient
Numerical Libraries

Fortran has been widely used by scientists and engineers for many years and therefore many algorithms to use in numerical calculations already exist.

These have been collected in a number of numerical libraries, some open (e.g. SLATEC [http://www.netlib.org/slatec/] and Numerical Recipes [http://www.nr.com/]) and some that cost (e.g. NAG [http://www.nag.co.uk]).
Some F90 Features
F90 has many familiar constructs:

- IF (expr) ...
The Constructs

F90 has many familiar constructs:

- IF (expr) ...
- IF (expr) THEN
  ...
  END IF
The Constructs

F90 has many familiar constructs:

- IF (expr) ...
- IF (expr) THEN
  ...
  END IF
- DO i = 1, n
  ...
  END DO
F90 has many familiar constructs:

- IF (expr) ...
- IF (expr) THEN
  ...
  END IF
- DO i = 1, n
  ...
  END DO
- Other forms of the DO construct
The Constructs

F90 has many familiar constructs:

- IF (expr) ...
- IF (expr) THEN
  ...
  END IF
- DO i = 1, n
  ...
  END DO
- Other forms of the DO construct
- CASE
Numeric Kind Parameterisation

Program test_kind
Implicit none
Real :: a
! selected_real_kind([p],[,r]) p = precision, r = range
Integer, parameter :: long = selected_real_kind(9,99)
Real(long) :: b

a = 1.7; b = 1.7_long

Print *, a,kind(a), precision(a), range(a)

Print *, b,kind(b), precision(b), range(b)

b = 1.7; print *, b
b = 1.7D0; print *,b

End Program test_kind
**Strong typing**: all typed entities must have their types specified explicitly.

By default an entity in Fortran that has not been assigned a type is *implicitly typed*, e.g. entities that begin with i,j,... are of type integer → **dangerous source of errors**

(Legend has it that error of this type caused the crash of the American Space Shuttle)

The statement **IMPLICIT NONE** turns on strong typing and its use is strongly recommended.
MODULE constants
  IMPLICIT NONE

  INTEGER, PARAMETER :: long = SELECTED_REAL_KIND(15, 307)
  REAL(long), PARAMETER :: pi = 3.14159265358979324D0
END MODULE constants

PROGRAM module_example
  USE constants
  IMPLICIT NONE

  REAL(long) :: a

  a = 2D0*pi
  print*, a
END PROGRAM module_example
MODULE circle
    USE constants
    IMPLICIT NONE

CONTAINS

    FUNCTION area(r)
        REAL(long), INTENT(IN) :: r
        REAL(long) :: area
        area = 2D0*pi*r
    END FUNCTION area

    FUNCTION circumference(r)
        REAL(long), INTENT(IN) :: r
        REAL(long) :: circumference
        circumference = pi*r**2
    END FUNCTION circumference

END MODULE circle
PROGRAM module_example2
    USE constants
    USE circle
    IMPLICIT NONE

    REAL(long) :: r, A, C

    r = 2
    A = area(r)
    C = circumference(r)

    print*, A, C

END PROGRAM module_example2
PROGRAM array
  USE constants
  IMPLICIT NONE

  REAL(long), DIMENSION(10,10) :: a
  REAL(long), DIMENSION(5,5) :: b, c
  REAL(long) :: d

  a = 1D0; b = 2D0

  c = MATMUL(a(1:5,6:10),b)
  c = c + b

  d = SUM(c)
  print*, d

END PROGRAM array
External Subroutines

SUBROUTINE area_rectangle(l, b, A)
    USE constants
    IMPLICIT NONE

    REAL(long), DIMENSION(:, :), INTENT(IN) :: l, b
    REAL(long), DIMENSION(size(l,1), size(l,2)) :: A

    A = l*b

END SUBROUTINE area_rectangle
External Subroutines - cont.

PROGRAM subr_example
  USE constants
  IMPLICIT NONE

  INTERFACE
    SUBROUTINE area_rectangle(l,b,A)
      USE constants
      IMPLICIT NONE
      REAL(long), DIMENSION(:,:), INTENT(IN) :: l,b
      REAL(long), DIMENSION(size(l,1), size(l,2)), INTENT(OUT) :: A
    END SUBROUTINE area_rectangle
  END INTERFACE

  REAL(long), DIMENSION(2,2) :: l,b,A
  l = 1D0; b = 2D0

  CALL area_rectangle(l,b,A); print*, A

END PROGRAM subr_example
External Subroutines - cont.

External subroutines are *implicitly* interfaced while module subroutines are *explicitly* interfaced.
External Subroutines - cont.

- External subroutines are *implicitly* interfaced while module subroutines are *explicitly* interfaced.
- External subroutines can be made explicitly interfaced by the use of an *interface block*.
External Subroutines - cont.

External subroutines are *implicitly* interfaced while module subroutines are *explicitly* interfaced.

External subroutines can be made explicitly interfaced by the use of an *interface block*.

Grouping related procedures and parameters into modules is good programming.
External Subroutines - cont.

- External subroutines are *implicitly* interfaced while module subroutines are *explicitly* interfaced.
- External subroutines can be made explicitly interfaced by the use of an **interface block**.
- Grouping related procedures and parameters into modules is good programming.

*We imagine subprogram libraries being written as sets of external subprograms together with modules holding interface blocks for them.*  
Metcalf & Reid
Summary
Summary

- Fortran has from the beginning been designed for numerical calculations
- The Fortran 90 standard modernised the language
- Array features make F90 especially attracting for numerical work
- Fortran is fast
Resources


- The Liverpool Fortran 90 courses homepage
  [http://www.liv.ac.uk/HPC/F90page.html](http://www.liv.ac.uk/HPC/F90page.html)


- [dbforums.lang.fortran](http://dbforums.com/f132/)