# Fortran 90/95 and Computational Physics

Jens Hjörleifur Bárðarson

jensba@raunvis.hi.is

University of Iceland

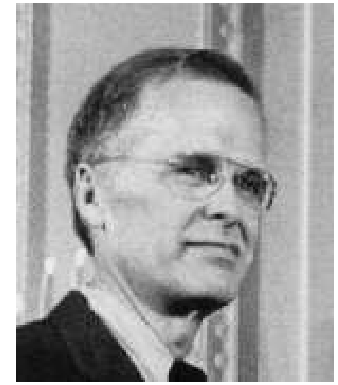
# Overview

- What is Fortran?
- Why Fortran?
- Some Important Things
- Summary

# What is Fortran 90?

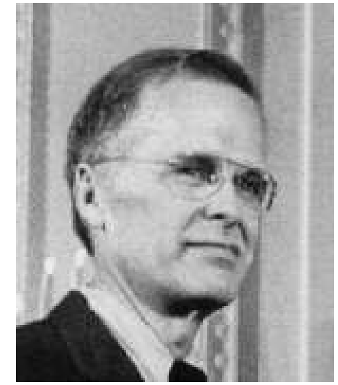
# The Origin

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.

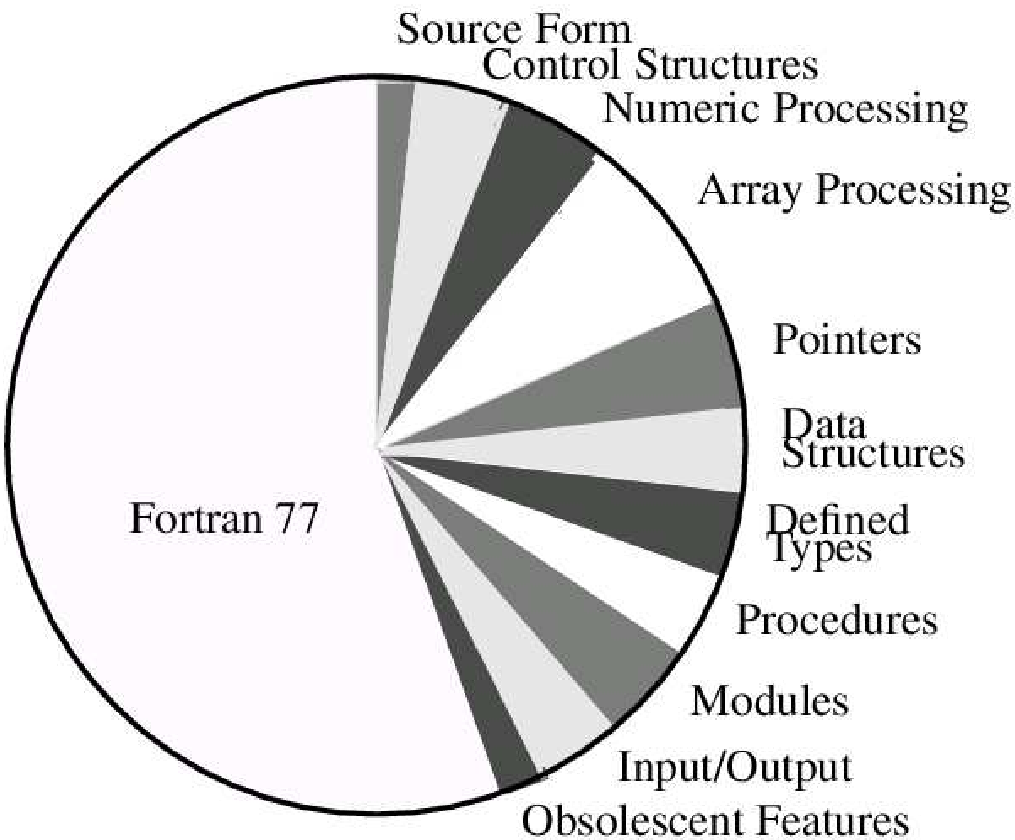


1966: The first ever standard for a programming language: Fortran 66

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?

| functionality | F77 | C | C++ | F90 |
|---|---|---|---|---|
| numerical robustness | 2 | 4 | 3 | 1 |
| data parallelism | 3 | 3 | 3 | 1 |
| data abstraction | 4 | 3 | 2 | 1 |
| object oriented programming | 4 | 3 | 1 | 2 |
| functional programming | 4 | 3 | 2 | 1 |
| average | 3.4 | 3.2 | 2.2 | 1.2 |

`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 scientist and engineers for many years and therfore many algorithms to use in numerical calculations already exist.

These have been collected in 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

# The Constructs

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

# 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

# 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
```

# IMPLICIT NONE

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 $\rightarrow$ 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

# Modules - Simple Example

```
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
```

# Modules - Another 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
```

# Modules - Another Example - cont.

```
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
```

# Array Features

```
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


- CSEP. Fortran 90 and Computational Science. Technical report, Oak Ridge National Laboratory, 1994

  `http://csep1.phy.ornl.gov/CSEP/PL/PL.html`
- The Liverpool Fortran 90 courses homepage

  `http://www.liv.ac.uk/HPC/F90page.html`
- Michael Metcalf and John Reid. *Fortran 90/95 explained*, second edition. Oxford, 1999
- Chivers and Sleightholme. *Introducing Fortran 95*. Springer, 2000
- Brainerd, Goldberg and Adams. *Programmer's Guide to Fortran 90*, third edition. Springer, 1996
- dbforums.lang.fortran `http://dbforums.com/f132/`