Introduction to Fortran 90

Abishek Gopal

TAMU Oceanography & HPRC

Texas A&M HPRC Short Course

November 12, 2021

TEXAS A&M UNIVERSITY Oceanography ĀĮM | **DIVISION OF RESEARCH** TEXAS A&M UNIVERSITY

Slides adapted from Jian Tao's Spring 21 Fortran course

Section 1

Introduction

Hello World in Fortran

Source code helloworld.f90

```fortran
program hello
  print *, 'Hello World!'
end program hello
```

Compile and run

```
$gfortran -o helloworld helloworld.f90
$./helloworld
Hello World!
```

What is Fortran?

- Fortran (formerly FORTRAN, derived from **FOR**mula **TRAN**slation) is a general-purpose, imperative programming language that is especially suited to numeric computation and scientific computing.
- Originally developed by IBM in the 1950s for scientific and engineering applications.
- Widely used in computationally intensive areas such as numerical weather prediction, finite element analysis, etc.
- It has been a popular language for high-performance computing and is used for programs that benchmark and rank the world's fastest supercomputers.

Why still learn Fortran?

- Well suited for scientific computing - A majority of scientific codes use Fortran.
- Second to none in terms of execution speed. Performance comparable to C.
- Very convenient array handling for scientific codes
- Actively supported by Intel, GNU, PGI, etc
- Optimized numerical libraries available for Fast Fourier Transforms, Linear Algebra, etc
- Can scale on hundreds of thousands of cores with MPI + OpenMP parallelization

History of Fortran

- FORTRAN — first released by IBM in 1956
- FORTRAN II — released by IBM in 1958
- FORTRAN IV — released in 1962, standardized
- FORTRAN 66 — appeared in 1966 as an ANSI standard
- FORTRAN 77 — appeared in 1977, structured features
- **Fortran 90** — 1992 ANSI standard, free form, modules
- **Fortran 95** — a few extensions
- Fortran 2003 — object oriented programming
- Fortran 2008 — a few extensions

The correct spelling of Fortran for 1992 ANSI standard and later (sometimes called Modern Fortran) is "Fortran". Older standards are spelled as "FORTRAN".

Section 2

Basics of Fortran 90 Language

Hardware and Software

A computing system is built from hardware and software

The hardware

is the physical medium and contains CPU, memory, keyboard, display, disks, ethernet interfaces etc.

The software

is a set of computer programs and contains operating system, compilers, editors, Fortran programs, etc.

Fortran Compiler

Compiler - from Wikipedia

A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as object code. The most common reason for converting source code is to create an executable program.

The FORTRAN team led by John Backus at IBM introduced the first unambiguously complete compiler in 1957.

Some popular Fortran compilers

GNU Fortran(gfortran), Intel Fortran(ifort), G95(g95), IBM(xlf90), Cray(ftn), Portland Group Fortran (pgf90)

Program - I

Typical Fortran program structure

```
PROGRAM program-name
IMPLICIT NONE
[specification part]
[execution part]
[subprogram part]
END PROGRAM program-name
```

The **PROGRAM** statement (optional) gives a name to the program. The first character of the name must be a letter. Use the **IMPLICIT NONE** statement to avoid implicit typing rules. The **END** statement terminates the program and returns control to the computer's operating system.

A Basic Fortran Example

Square of a number

```fortran
PROGRAM square          Program declaration
! Comment
IMPLICIT NONE
REAL :: x, x_sq         Variable declaration

WRITE (*,*) 'Enter the value of x:'

READ (*,*) x            Read x

y = x**2                Assignment

WRITE (*,*) 'The square of x is:', y

END PROGRAM square
```

Fortran Source Code

- Fortran 90 and later versions support free format source code.
- Fortran source code is in ASCII text and can be written in any text editor.
- Fortran source code is case **insensitive**. PROGRAM is the same as Program and pRoGrAm.
- Use whatever convention you are comfortable with and be consistent throughout.
- Comments in Fortran 90 source code start with an exclamation mark (!) except in a character string. Comments help to enhance the readability of your code.

Statements

A statement is a complete instruction. Statements may be classified into two types: executable and non-executable.

- Executable statements are those which are executed at runtime.
- Non-executale statements provide information to compilers.
- If a statement is too long, it may be continued by the ending the line with an **ampersand (&)**.
- Max number of characters (including spaces) in a line is 132 though it's standard practice to have a line with up to 80
- Multiple statements can be written on the same line provided the statements are separated by a semicolon.

Variables

Variables are the fundamental building blocks of any program

- A variable name may consist of up to 31 alphanumeric characters and underscores, of which the first character must be a letter.
- There are no reserved words in Fortran.
- Variable names must begin with a letter and should not contain a space.

Variable Types

Intrinsic data types

- INTEGER: exact whole numbers
- REAL: real, franctional numbers
- COMPLEX: complex, fractional numbers
- LOGICAL: boolean values
- CHARACTER: strings

- Users can define additional types.
- REAL is a single-precision floating-point number.
- FORTRAN provides DOUBLE PRECISION data type for double precision REAL. This is obsolete but is still found in many programs.

Contants - I

Integer

```
242, -2341, 290223
```

Real (single precision)

```
1.03, 3.51e23, -8.201
```

Real (double precision)

```
1.03d0, 3.51d23, -8.201d0
```

Constants - II

Complex (single precision)

```
(1.0,0.0), (-2.5e-5, 3.0e-6)
```

Complex (double precision)

```
(1.0d0,0.0d0), (-2.5d-5, 3.0d-6)
```

Logical

```
.True., .False.
```

Character

```
"Hello World!", "Is pi 3.1415926?"
```

Explicit and Implicit Typing

Implicit typing of variables

$$\underbrace{ABCDEFGH}_{real} \overbrace{IJKLMN}^{integer} \underbrace{OPQRSTUVWXYZ}_{real}$$

`IMPLICIT DOUBLE PRECISION (a-h,o-z)`

- it is highly recommended to explicitly declare all variable and avoid implict typing using the statement.

    `IMPLICIT NONE`

- the IMPLICIT statement must precede all variable declarations.

Variable Declarations - I

Numerical variables

```fortran
INTEGER :: i, j = 2
REAL :: a, b = 4.d0
COMPLEX :: x, y
```

Constant variables

```fortran
INTEGER, PARAMETER :: j = 2
REAL, PARAMETER :: pi = 3.14159265
COMPLEX, PARAMETER :: ci = (0.d0,1.d0)
```

Variable Declarations - II

Logical variables

```fortran
LOGICAL :: l, flag=.true.
```

Character variables

The length of a character variable is set with **LEN**, which is the maximum number of characters (including space) the variable will store. By default, **LEN=1** thus only the first character is saved in memory if **LEN** is not specified.

```fortran
CHARACTER(LEN=10) :: a
CHARACTER :: ans = 'yes' !stored as ans='y'
```

KIND Parameter for Variables

- Fortran 90 introduced **KIND** parameters to parameterize the selection of different possible machine representations for each intrinsic data types.
- The **KIND** parameter is an integer which is processor dependent.
- There are only 2(or 3) kinds of reals: 4-byte, 8-byte (and 16-byte), respectively known as single, double (and quadruple) precision.
- The corresponding **KIND** numbers are 4, 8 and 16 for most compilers.

Operators

Arithmetic Operators

`+ , - , * , / , **`

Relational Operators

`== , <. <= , > , >= , /=`

Logical Operators

`.AND. , .OR. , .NOT. , .EQV. , .NEQV.`

Character Concatenation Operator

`//`

Operator Precedence

- All operator evaluations on variables is carried out from left-to-right.
- Arithmetic operators have a highest precedence while logical operators have the lowest precedence
- The order of operator precedence can be changed using parenthesis, '(' and ')'
- Users can define their own operators.
- Extra parenthesis could be added to enhance readability and avoid mistakes.

Expressions

An expression is a combination of one or more operands, zero or more operators, and zero or more pairs of parentheses.

Arithmetic expressions

```
y + 1.0 - x, sin(x) + y
```

Relational expressions

```
a .and. b, c .neqv. d
```

Character expressions

```
'hello' // 'world', 'ab' // 'xy'
```

Intrinsic Functions - I

Fortran provides many commonly used functions, called intrinsic functions.

Numerical functions

```
ABS(A), CEILING(A), FLOOR(A), MAX(A,B),
MIN(A,B), MOD(I,J), SQRT(A), EXP(A), LOG(A),
LOG10(A), INT(A), REAL(A), DBLE(A),
CMPLX(A[,B]), AIMAG(A)
```

Math functions

```
SIN(A), COS(A), TAN(A), ASIN(A),
ACOS(A), ATAN(A), ATAN2(A,B), SINH(A),
COSH(A), TANH(A)
```

Intrinsic Functions - II

Character functions

```
LEN(S), LEN_TRIM(S), LGE(S1,S2), LGT(S1,S2),
LLE(S1,S2), LLT(S1,S2), ADJUSTL(S),
ADJUSTR(S), REPEAT(S, N), SCAN(S, C), TRIM(S)
```

Array functions

```
SIZE(A[,N]), SUM(A[,N]), PRODUCT(A[,N]),
TRSNSPOSE(A), DOT_PRODUCT(A,B), MATMUL(A,B),
CONJG(X)
```

Exercise

Write a short Fortran program to convert 10 Celsius to Fahrenheit, and 40 Fahrenheit to Celsius.
The expressions to convert between the two are given below:

$$T(F) = \frac{9}{5}T(C) + 32$$

$$T(C) = \frac{5}{9}(T(F) - 32)$$

Section 3

Control Structures

Control Constructs

A Fortran program is executed sequentially. Control Constructs change the sequential execution order of the program.

- Conditionals: **IF**, **IF**-**THEN**-**ELSE**
- Switches: **SELECT/CASE**
- Loops: **DO**
- Branches: **GOTO** (obsolete in Fortran 95/2003, use CASE instead)

Conditionals

IF construct

```
if ( expression ) statement
```

IF THEN ELSE construct

```
if ( expression 1 ) then
  executable statements
else if ( expression 2 ) then
  executable statements
else
  executable statements
end if
```

Conditionals - IF Example

IF construct

```
if (value < 0) value = 0
```

- When the if statement is executed, the logical expression is evaluated.
- If the result is true, the statement following the logical expression is executed; otherwise, it is not executed.
- The statement following the logical expression cannot be another if statement. Use the if-then-else construct instead.

Conditionals – IF-THEN-ELSE Example

IF THEN ELSE construct

```fortran
if ( x < 50 ) then
  GRADE = 'F'
else if ( x >= 50 .and. x < 60 ) then
  GRADE = 'D'
else if ( x >= 60 .and. x < 70 ) then
  GRADE = 'C'
else if ( x >= 70 .and. x < 80 ) then
  GRADE = 'B'
else
  GRADE = 'A'
end if
```

Exercise

The roots of a quadratic equation $ax^2 + bx + c = 0$ are given by:

$$q1 = \frac{1}{2a}(-b + \sqrt{b^2 - 4ac})$$

$$q2 = \frac{1}{2a}(-b - \sqrt{b^2 - 4ac})$$

Write a Fortran program to read in (or assume) the coefficients a, b and c, and then compute the roots of the corresponding equation. Using an **if clause**, check that the discriminant $b^2 - 4 * a * c$ is positive before doing the computation, otherwise print that the roots are complex and exit.

Switches

SELECT CASE construct

```
[case_name:] select case ( expression )
  case ( selector )
    executable statement
  case ( selector )
    executable statement
  case default
    executable statement
end select [case_name]
```

The value of the expression in the select case should be an integer or a character string. The case name is optional.

Switches - Example I

Character case selector

```
select case ( traffic_light )
  case ( "red" )
    print *, "Stop"
  case ( "yellow" )
    print *, "Caution"
  case ( "green" )
    print *, "Go"
  case default
    print *, "Illegal value: ", traffic_light
end select
```

Switches - Example II

Integer case selector

```fortran
select case ( score )
  case ( 50 : 59 )
    GRADE = "D"
  case ( 60 : 69 )
    GRADE = "C"
  case ( 70 : 79 )
    GRADE = "B"
  case ( 80 : )
    GRADE = "A"
  case default
    GRADE = "F"
end select
```

Loops - DO

DO construct

```
[do name :] do loop_control
  execution statements
end do [do name]
```

The do loop name is optional. To exit the do loop, use the **EXIT** or **CYCLE** statement.

- The **EXIT** statement causes termination of execution of a loop.
- The **CYCLE** statement causes termination of the execution of one iteration of a loop.

Loops - DO Example

Factorial with DO construct

```fortran
program factorial1
  implicit none
  integer(KIND=8) :: i,factorial, n=6
  factorial = n
  do i = n-1,1,-1
    factorial = factorial * i
  end do
  write(*,'(i4,a,i15)') n,'!=',factorial
end program factorial1
```

Loops - DO WHILE

If a condition is to be tested at the top of a loop, a do ... while loop can be used

DO WHILE construct

```
[do name:] do while ( expression )
  executable statements
end do [do name]
```

The loop only executes if the logical expression is **.TRUE.**

Loops - DO WHILE Example

DO WHILE example

```
finite: do while ( i <= 100 )
  i = i + 1
  inner: if ( i < 10 ) then
    print *, i
  end if inner
end do finite
```

Exercise

We know that the summation of the first n positive integers is given by the arithemtic sequence formula

$$1 + 2 + 3 + ... + n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Write a Fortran program that uses a **do loop** to compute the sum of the first n integers. You may assume a value of **n**, or read in as input. After computing the sum, use an **if clause** to compare the computed sum with the expected value (RHS), and print an affirmation to the output.

15 minute break

Section 4

Program Sub-units

Program - I

Typical Fortran program structure

```
PROGRAM program-name
  IMPLICIT NONE
  [specification part]
  [execution part]
  [subprogram part]
END PROGRAM program-name
```

The **PROGRAM** statement (optional) gives a name to the program. The first character of the name must be a letter. Use the **IMPLICIT NONE** statement to avoid implicit typing rules. The **END** statement terminates the program and returns control to the computer's operating system.

Structured Programming

A Fortran program can consist of one or more program sub-units
units, **SUBROUTINE, FUNCTION, MODULE**. Why?

- Break down a complex program into smaller, coherent units
- To avoid repeating code blocks
- Allow "Unit" testing and debugging
- Enhance code readability

An Example

Temperature Conversion between Fahrenheit and Celsius

```fortran
program temp
  implicit none
  real :: tempC, tempF
! Convert 10C to fahrenheit
  tempF = 9.0 / 5.0 * 10.0 + 32.0
! Convert 40F to celsius
  tempC = 5.0 / 9.0 * (40.0 - 32.0 )
  call display(tempc, tempF)
end program temp
```

Subroutines - I

Typical Fortran subroutine structure

```fortran
SUBROUTINE subroutine-name(dummy arguments)
  IMPLICIT NONE
  [specification part]
  [execution part]
  [subprogram part]
END SUBROUTINE subroutine-name
```

Subroutines - II

- CALL Statement:
    - The **CALL** statement evaluates its arguments and transfers control to the subroutine
    - Upon return, the next statement is executed.
- SUBROUTINE Statement:
    - The **SUBROUTINE** statement declares the procedure and its arguments.
    - These are also known as dummy arguments.
- The subroutine's interface is defined by
    - The subroutine statement itself
    - The declarations of its dummy arguments
    - Anything else that the subroutine uses

A Subroutine Example

Calcuate the sum of input variables

```fortran
SUBROUTINE calc(a,b,c)
  IMPLICIT NONE
  real :: a,b,c
  c = a + b
  return
END SUBROUTINE calc
```

Functions

Fortran functions operate on the same principle as subroutines. The only difference is that function returns a value and does not involve the call statement.

Calculate the sum of the input variables

```fortran
FUNCTION calc(a,b)
  IMPLICIT NONE
  real :: a,b,calc
  calc = a + b
END FUNCTION calc
```

Sum of two real numbers

Subroutine definition

```
SUBROUTINE calc(a,b,c)
 IMPLICIT NONE
 real :: a,b,c
 c = a + b
END SUBROUTINE calc
```

Function definition

```
FUNCTION calc(a,b)
 IMPLICIT NONE
 real :: a,b,calc
 calc = a + b
END FUNCTION calc
```

Subroutine call

```
CALL calc(x,y,z)
```

Function call

```
 z = calc(x,y)
```

Recursive Functions

In Fortran 90, recursive procedures call themselves and must be declared explicitly. The recursive function declarations must contain a result keyword, and one type of declaration refers to both the function name and the result variable.

Calcuate the factorial

```
RECURSIVE FUNCTION fact(i) result(i_fact)
integer :: i, i_fact
if ( i > 0 ) then
  i_fact = i * fact(i - 1)
else
  i_fact = 1
end if
END FUNCTION fact
```

Modules

A module is a program unit whose functionality can be exploited by other programs which attaches to it via the **USE** statement.

```fortran
MODULE mymod
IMPLICIT NONE
integer , parameter :: dp=8
CONTAINS

 FUNCTION calc(a,b)
  IMPLICIT NONE
  real :: a,b,calc
  calc = a + b
 END FUNCTION calc
END MODULE mymod
```

Module invocation

```fortran
PROGRAM test
 USE mymod
 IMPLICIT NONE
 REAL(KIND=dp)::x,y,z
 z = calc(x,y)
END PROGRAM test
```

Exercise

We already computed the roots of a quadratic equation as:

$$q = \frac{1}{2a}(-b \pm \sqrt{b^2 - 4ac})$$

Convert your previous Fortran program to use subroutines. The main program should read in (or assume) the coefficients a, b and c, and then call the subroutine as:

```fortran
CALL quad_roots(a,b,c, q1, q2, is_complex)
```

where q1 and q2 are the output roots of type REAL, and **is_complex** is an output boolean variable that specifies if roots are real or complex.

Section 5

Input and Output in Fortran

Simple I/O

Any program needs to be able to read input and write output to be useful and portable.

Simple output with PRINT

```
print *, <var1> [, <var2> [, ... ]]
```

Simple input with READ

```
read *, <var1> [, <var2> [, ... ]]
```

The * indicates that the data is unformatted.

Example with Simple I/O

Interactive hello world via I/O

```fortran
PROGRAM hello
  IMPLICIT NONE
  character(len=100) :: your_name
  print *, 'Your Name Please'
  read *, your_name
  print *, 'Hello ', your_name
END PROGRAM hello
```

I/O with Unit Number

Files are identified by some form of file handle, in Fortran called the unit number.

- The default unit number 5 is associated with the standard input,
- Unit number 6 is assigned to standard output.

Read and write through unit number

```
read(unit,*)
write(unit,*)
```

File Operations - I

Fortran provides functions to open, read, write, inquire, and close files. A file may be opened with the statement

```
OPEN ([UNIT=]un, FILE=fname [, options])
READ (un, options) varlist
WRITE (un, options) varlist
INQUIRE ([UNIT=]un, options)
CLOSE ([UNIT=]un [, options])
```

File Operations - II

If data is read/written from/to standard input/output, then

- the unit number can also be replaced with *.
- use alternate form for reading and writing i.e. the READ *, and PRINT start mentioned earlier.
- If data is unformatted i.e. plain ASCII characters, the option to WRITE and READ command is *.

Formatted I/O

A formatted data description must adhere to the generic form:
nCw.d

- n is an integer constant that specifies the number of repititions (default 1 can be omitted),
- C is a letter indicating the type of the data variable to be written or read,
- w is the total number of spaces allocated to this variable, and,
- d is the number of spaces allocated to the fractional part of the variable. Integers are padded with zeros for a total width of w provided $d \leq w$.
- The decimal (.) and d designator are not used for integers, characters or logical data types.

FORMAT Statement - I

In the simplest form, the format is enclosed in single quotes and parentheses as argument to the keyword.

```fortran
print '(I5,5F12.6)', i, a, b, c, z ! complex z
write(6, '(2E15.8)') arr1, arr2
read(5, '(2a)') firstname, lastname
```

If the same format is to be used repeatedly or it is complicated, the FORMAT statement can be used. The **FORMAT** statement must be labeled and the label is used in the input/output statement to reference it

```fortran
label FORMAT(formlist)
PRINT label, varlist
WRITE(un, label) varlist
READ(un, label) varlist
```

FORMAT Statement - II

The **FORMAT** statements can occur anywhere in the same program unit. Most programmers list all **FORMAT** statements immediately after the type declarations before any executable statements.

Format statement examples

```
10 FORMAT(I5,5F12.6)
20 FORMAT(2E15.8)
100 FORMAT(2a)
print 10, i, a, b, c, z ! complex z
write(6,20) arr1, arr2
read(5,100) firstname, lastname
```

Section 6

Array Handling

Array Declarations

Arrays (or matrices) hold a collection of different values at the same time. Individual elements are accessed by subscripting the array. Fortran arrays are defined with the keyword **DIMENSION(lower bound: upper bound)**
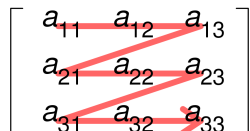
Arrays

```fortran
INTEGER, DIMENSION(1:106) :: atomic_number
REAL, DIMENSION(3, 0:5, -10:10) :: values
CHARACTER(LEN=3), DIMENSION(12) :: months
```
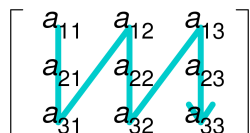
In Fortran, arrays can have up to seven dimensions. Fortran arrays are column major.

Row Major vs Column Major

Row-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Column-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

In Fortran

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{13} \\ a_{23} \\ a_{33} \end{bmatrix}$$

Arrays in use

$$y = Ax = \sum_{j=1}^{N} a_j x_j$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

```
INTEGER :: m = 3, n = 4, i, j
REAL :: A(m,n), x(n), y(m)

y=0.0
do j = 1, n
y(:) = y(:) + A(:,j)*x(j)
end do
```

Section 7

Online Resources for Fortran

Program - I

- Numerical Recipes - http://numerical.recipes/
- Lahey Fortran Resources - http://www.lahey.com/other.htm

Section 8

Hands-on Exercise