Introduction to Fortran

Drew Schmidt

January 16, 2014
Contents

1 Introduction
2 Basics
3 Control
4 Functions, Intrinsics, and Subroutines
5 Precision
6 Wrapup
Contents

1 Introduction
   • Background
   • Hello World
What is Fortran?

- Formula translation system.
- General purpose programming language.
- Well-suited for mathematics and engineering.
- Compiled (rather than interpreted)
- Portable.
Created at IBM in 1954 as an alternative to assembly language.

Debuted in the days of punch cards.

Regularly gets updated to a new standard.
What people think Fortran is like

```
1121 FORMAT(I4,F8.3)
3298 CONTINUE
IF(MOD(I,A)$.EQ.Z) THEN
  GOTO 2359
ELSE IF(MOD(I,B)$.EQ.Z) THEN
  GOTO 8125
ELSE
  WRITE (*,2930) I
  GOTO 7365
END IF
7235 FORMAT(A,F5.3)
7356 CONTINUE
I=I+$+1
GOTO 1249
2930 FORMAT(I4,$)
2359 CONTINUE
```
Standards

Fortran standards are denoted by year.

- Fortran 66
- Fortran 77
- Fortran 90
- Fortran 95
- Fortran 2003
- Fortran 2008
- Fortran 2015
Language Features

Fortran essentially has two formats: Fortran 77 and “modern”. All versions have:

- Matrices.
- Complex numbers.
- Good interoperability with C.
- REALLY good compilers.

Modern variants of Fortran additionally have:

- Dynamic memory allocation.
- Pointers.
- OOP.
Applications for Fortran

Bad applications:
- A website
- An OS kernel
- Text processing
- Random number generators

Good applications:
- Mathematics
- Engineering
- Statistics
- Scientific computing
Compilers

- GNU gfortran (free)
- Intel ifortran
- Portland Group pgfortran
- ...
Why Fortran?

- High level language
- Avoids fiddly memory management like in C
- Fortran binaries are VERY fast
- Good HPC support (MPI, OpenMP, . . .)
Hello world example

**hello.f**

```
1  program  helloworld
2  print  *,  "Hello world"
3  end  program  helloworld
```
Hello World

Hello world example

hello.f90

```fortran
program helloworld
  print *, "Hello world"
end program helloworld
```
Comments

- .f is for F77, .f90 is for F90+
- Don’t use F77.
- Fortran is case insensitive.
- print and write print/write output
- read reads input
Contents

2 Basics
- Numeric Types
- Logical Type
Basic Type

- Integer
- Real/double
- Complex/double complex
- Logical
- Character
## Numeric Types

### Integer

Whole numbers.

- 1
- -517
- 0
- Not 1.2
- Not \(\pi\)
- Not 0.0
Real and Double

Floating point

- 1.1
- 3.14159
- Not 1
- Not $\sqrt{-1}$
# Basic Numeric Operations

- **a=b:** assign the value of b to a
- **a+b:** add a and b
- **a−b:** subtract b from a
- **a*b:** multiply a and b
- **a/b:** divide b into a
- **a**\(^**b**\): raise a to the power b
Quick example

```fortran
program arithmetic
  integer :: a = 2, b = 3

  print *, a+b
  print *, a**b
  print *, a/b
end program
```

5
8
0
Logical

Logical variables can take two values:

- .true.
- .false.
Comparing Logicals

- `.eqv.` tests if two logical expressions are equivalent
- `.neqv.` tests if two logical expressions are *not* equivalent
- `.and.` the and operator
- `.or.` the or operator
- `.not.` the negation operator
Comparing Numerics

- \( a < b \) or \( a . l t . b \): \( a \) less than \( b \)
- \( a \leq b \) or \( a . l e . b \): \( a \) less than or equal to \( b \)
- \( a > b \) or \( a . g t . b \): \( a \) greater than \( b \)
- \( a \geq b \) or \( a . g e . b \): \( a \) greater than or equal to \( b \)
- \( a = b \) or \( a . e q . b \): \( a \) equal to \( b \)
- \( a /= b \) or \( a . n e . b \): \( a \) not equal to \( b \)

The type of a numeric comparison is of type \textit{logical}. 
Comparing Numerics

Note: The output of a comparison of numerics is logical:

- $a < b < c$ makes no sense (types mismatch)
- Instead: $a < b \ . \ and \ . \ b < c$
Introduction
Basics
Control
Functions
Precision
Wrapup

Logical Type

Quick example

```fortran
program logica1s
    integer :: a = 2, b = 3, c = 1

    print *, a < b
    print *, a /= b .and. a < c
end program
```

T
F
Implicit Declaration

- In Fortran, variables may be used *implicitly*.
- Do not get into the habit of doing this.
- You can turn this off in a program (function, subroutine) by declaring *implicit none*.
- Declaring *implicit none* is generally recommended.
Implicit variables quick example

Compiles:

```fortran
program implicit_declaration
  a = 2
  b = 3
  print *, a+b
end program
```

Fails to compile:

```fortran
program implicit_declaration
  implicit none
  a = 2
  b = 3
  print *, a+b
end program
```
Logical Type

Example

Go to example.

Not covered:
- complex
- character
- kind/precision
Contents

3 Control
  - if-then-else
  - Loops
if-then-else

Conditionals can take a few different forms:

1. `if (condition) one-liner`

2. ```
   if (condition 1) then
       statement
   else if (condition 2) then
       statement
   else if (...) then
       ...
   else
       statement
   end if
   ```

Note: the `else if` and `else` pieces are optional
if-then-else quick example

```fortran
integer :: score
character(len=1) :: grade

if (score < 60) then
  grade = "F"
else if (score < 70) then
  grade = "D"
else if (score < 80) then
  grade = "C"
else if (score < 90) then
  grade = "B"
else
  grade = "A"
end if
```
do Loops

```fortran
1 do index = first, last, step
2     ! statements
3 end do
```

- index, first, last are integers
- step is a non-zero integer
- step can be omitted (and in this case, the step is 1)
**Do loops quick example**

```fortran
integer :: factorial, i, n

print *, "Give me a positive integer integer :"
read *, n

factorial = 1

do i = 2, n
    factorial = factorial * i
end do

print *, n, "! = ", factorial
```
do Loops

1. do
2.   ! statements
3. end do

- statements are executed repeatedly
- To exit the loop, use exit
- To jump to the next iteration, use cycle
do loops quick example

```fortran
integer :: f, i, n

print *, "Give me an integer:"
read *, n

f = 1
i = 2
do
  if (i <= n) then
    f = f * i
    i = i + 1
  else
    exit
  end if
end do

print *, n, "! = ", f
```
Loops

Example

Go to example.

Not covered: do-while loops
Contents

4 Functions, Intrinsics, and Subroutines
- Functions
- Intrinsics
- Subroutines
Functions

1. Declaration
2. `function foo(bar)
   3.   type :: foo
   4.   ! statements
   5. end function

6. Invocation
7. `a = foo(b)

- Can take variety of inputs.
- Returns single output.
Functions quick example

```fortran
function circumference(r)
    implicit none
    real :: pi = 3.14159
    real :: r
    real :: circumference
    circumference = 2.0*pi*r
end function circumference

program circles
    real :: r = 2.0
    real :: circumference
    print *, circumference(r)
end program circles
```

12.5663605
Intrinsics

- Built-in functions.
- Casting.
- Basic math utilities.
- Bit-shifting.
# Intrinsic Examples

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Effect</th>
<th>Intrinsic</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Convert to integer</td>
<td>mod</td>
<td>Modular arithmetic</td>
</tr>
<tr>
<td>real</td>
<td>Convert to real</td>
<td>abs</td>
<td>Absolute value</td>
</tr>
<tr>
<td>floor</td>
<td>Greatest integer below</td>
<td>sqrt</td>
<td>Square root</td>
</tr>
<tr>
<td>ceiling</td>
<td>Smallest integer above</td>
<td>exp</td>
<td>Exponential</td>
</tr>
</tbody>
</table>
Intrinsics quick example

```
integer :: a = 2, b = 3
print *, mod(3, 2)
pick * , real(a/b)
pick * , real(a) / real(b)
```

```
1
0.00000000
0.666666687
```
Subroutines

! Declaration
subroutine foo (bar, baz)
    type :: bar, baz
    ! statements
end subroutine

! Invocation
call foo (a, b)

- Can take variety of inputs.
- Returns a variety of outputs (modifying values of inputs).
- Equivalent in C is void function.
Intent

- Can declare intention of use for variable.
- `intent(in)`, `intent(out)`, `intent(inout)`.
- Like `implicit none`, not strictly necessary, but useful.

```fortran
subroutine foo(a, b)
  integer, intent(in) :: a
  integer, intent(out) :: b
  ! statements
end subroutine
```
Subroutine quick example

```fortran
subroutine circ_stuff(r, circumference, area)
   implicit none
   real :: pi = 3.14159
   real, intent(in) :: r
   real, intent(out) :: circumference, area
   circumference = 2.0 * pi * r
   area = pi * r * r
end subroutine circ_stuff

program circles
   real :: r = 2.0, circumference, area
   call circ_stuff(r, circumference, area)
   print *, "Circumference = ", circumference
   print *, "Area = ", area
end program circles
```

Circumference = 12.5663605
Area = 12.5663605
Example

Go to example.
Contents

5 Precision
   - Kind
In C, there are two floating point types

- float
- double

In elder Fortran, the corresponding types are

- real
- double precision
Scientific Notation and Precision

\[
\text{number} = \text{mantissa} \times 10^{\text{exponent}}
\]

\[1.234 = 1234 \times 10^{-4}\]

- **real**: 23 bit of mantissa, 8 bits of exponent, and 1 sign bit.
- **double precision**: 52 bit of mantissa, 11 bits of exponent, and 1 sign bit.

- **real**: \(\approx 6\) decimal digits of precision
- **double precision**: \(\approx 15\) decimal digits of precision
Kind

- kind is a parameter that specifies the storage/precision of a type (beyond its default)
- real(kind=4)
- Different compilers assign different meaning to kind.
- Avoid this complexity with the intrinsic functions selected_<type>_kind
  - real(kind=selected_real_kind(6))
  - real(kind=selected_real_kind(15))
  - ...

Drew Schmidt
Introduction to Fortran
program kind_example
  implicit none
  integer, parameter :: r15 = selected_real_kind(15)
  real :: x
  real(kind=r15) :: y

  x = 1.0
  y = 1.0

  print *, x
  print *, y
end program

1.00000000
1.0000000000000000
Contents

6 Wrapup
Other Important Topics Not Discussed Here

- Debugging
- Arrays
- Pointers
- Interfacing to C
- Tabs versus spaces
Some languages come and go
But with Fortran...
Because when the USS Enterprise makes her maiden voyage
Beneath the fancy blinking screens
You can bet that their crucial systems are powered by Fortran written in the 1970's
Thanks for coming!

Questions?