Fortran

Dr. Hyrum Carroll

September 8 — 27, 2016

Computing Background

What operations can a computers do?

- Simple operations with two bytes (addition, subtraction, EOR, AND)
- Simple operations with one byte (bit changes, increment by one, shift left ...)
- Simple testing of one or two bytes (equal, high bit set, ...)
- ALL of these are doing with simple logic chips!

Memory

- Memory is based on storing computer bits
- There are many ways to do this
  - storing data on paper with holes or ink
  - storing electronic charge
  - preserving the state of an electronic device (a flip-flop)
  - putting data into a magnetically aligned area of a disk drive
  - storing data as 'pits' or 'lands' on a CD-ROM / DVD, and using phase differences in reflected light to optically read them
  - In all cases, the data can be accessed and stored using small currents and voltages, and at very high data rates
- Data formats media change, but the underlying data remains basically constant

Bytes

- We generally don't talk about individual bits, but we group them in 8-bit segments called Bytes
- Bytes of values from
  - 0 to 255 Decimal
  - 00000000 to 11111111 Binary
  - 00 to FF Hexadecimal
- We then group Bytes together in the form of Words
- Depending on the Chips and Operating systems, Words can be
  - 4 bytes long - 32 bits
  - 8 bytes long - 64 bits
  - rarely - 16 bytes long - 128 bits

Counting in Binary

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
16	10000
32	100000

Binary Numbers

Decimal	Binary
0	0
1	1
2	10
4	100
8	1000
16	10000
32	100000
37	100101
0.5	0.10000
0.25	0.01000
0.125	0.00100
0.0625	0.00010
0.03125	0.00001
0.65625	0.10101

More Binary Numbers

We can express 83.65625 as $83 + 0.65625$. In binary, this becomes $1010011.00 + 0.10101 = 1010011.10101$

we can also express this as:

1010011.10101×2^0 or
101001.110101×2^1 or
10100.1110101×2^2 or
1010.01110101×2^3 or
101.001110101×2^4 or
10.1001110101×2^5 or
1.01001110101×2^6

ASCII Data

(decimal = ASCII)

48 = 0	49 = 1	50 = 2	51 = 3
52 = 4	53 = 5	54 = 6	55 = 7
56 = 8	57 = 9	58 = :	59 = ;
60 = <	61 = =	62 = >	63 = ?
64 = @	65 = A	66 = B	67 = C
68 = D	69 = E	70 = F	71 = G
72 = H	73 = I	74 = J	75 = K
76 = L	77 = M	78 = N	79 = O
80 = P	81 = Q	82 = R	83 = S
84 = T	85 = U	86 = V	87 = W
88 = X	89 = Y	90 = Z	91 = [
92 = \	93 =]	94 = ^	95 = _
96 = `	97 = a	98 = b	99 = c
100 = d	101 = e	102 = f	103 = g
104 = h	105 = i	106 = j	107 = k
108 = l	109 = m	110 = n	111 = o

Computer Memory

The data string - "Hello World"

is transferred into memory by using computer Bytes and Words

	Byte			
Word	0	1	2	3
0	H	e	l	l
1	o		W	o
2	r	l	d	
3				

Computer Memory

	Byte			
Word	0	1	2	3
0	H	e	l	l
1	o		W	o
2	r	l	d	
3				

The characters are changed into numbers

	Byte			
Word	0	1	2	3
0	71	101	118	118
1	111	32	87	111
2	113	108	100	
3				

Computer Memory

	Byte			
Word	0	1	2	3
0	71	101	118	118
1	111	32	67	68
2	83	32	49	48
3	49	32	32	32

The numbers are stored as binary zeros and ones-

	Byte			
Word	0000	0001	0010	0011
000	0100 0111	0110 0101	0111 0110	0111 0110
001	0110 1111	0010 0000	0100 0011	0100 0100
010	0101 0011	0001 0000	0011 0001	0011 0000
011	0011 0001	0001 0000	0001 0000	0001 0000

A slightly different string

Data in Memory

	Byte			
Word	0000	0001	0010	0011
0000	0100 0111	0110 0101	0111 0110	0111 0110
0001	0110 1111	0010 0000	0100 0011	0100 0100
0010	0101 0011	0001 0000	0011 0001	0011 0000
0011	0011 0001	0001 0000	0001 0000	0001 0000
0100	0100 0111	0110 0101	0111 0110	0111 0110
0101	0110 1111	0010 0000	0100 0011	0100 0100
0110	0101 0011	0001 0000	0011 0001	0011 0000
0111	0011 0001	0001 0000	0001 0000	0001 0000
1000	0100 0111	0110 0101	0111 0110	0111 0110
1001	0110 1111	0010 0000	0100 0011	0100 0100
1010	0101 0011	0001 0000	0011 0001	0011 0000
1011	0011 0001	0001 0000	0001 0000	0001 0000
1100	0100 0111	0110 0101	0111 0110	0111 0110
1101	0110 1111	0010 0000	0100 0011	0100 0100
1110	0101 0011	0001 0000	0011 0001	0011 0000
1111	0011 0001	0001 0000	0001 0000	0001 0000

Fortran

A Very Simple Fortran Program

Using an editor, create a program called “helloWorld.f90”

```
program hello
! a comment — hello world program
print *, "Hello World!"
end program hello
```

A Very Simple Fortran Program

Compile and execute the program helloWorld.f90:

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

- The default executable name is "a.out" if you didn't use the `-o` flag to specify the executable name
- To execute a program, you need to specify its path or modify your PATH variable

A Very Simple Fortran Program

Some notes

- Fortran is **case insensitive** - capital and small letters are identical to the compiler except when they are in quotes
- **Comments** start with a '`!`'
- The ***** after the print statement indicates the **output goes to the screen** (aka STDOUT)
- print*, is an **unformatted print statement**
- The name of the program **does not need to match** the name of the f90 file or the final name of the executable
- Need to **use .f90** file extension if using `gfortran`
- Lines must be **less than 132 characters** unless a continuation character is used (add '/' at the end of the line)
- **Indenting** is **optional**
- "Consistently separating words by spaces became a general custom about the tenth century A. D., and lasted until about 1957, when FORTRAN abandoned the practice." (Sun FORTRAN Reference Manual)

Fortran Example

newfile.f90:

```
program newfile

! This is a comment and is in gray

                                          pRiNT*,'This our new file!'

end ProgRam nEwfIle
```

Non-Free Form and Free Form

```
$ cp helloWorld.f90  helloWorld.f
$ gfortran helloWorld.f  -o helloWorld-oldSchool
helloWorld.f:1.1:

program hello
 1
Error: Non-numeric character in statement label at (1)
helloWorld.f:1.1:

program hello
 1
Error: Unclassifiable statement at (1)
helloWorld.f:3.3:

  print *, "Hello World!"
   1
Error: Non-numeric character in statement label at (1)
...
```

Non-Free Form and Free Form

History

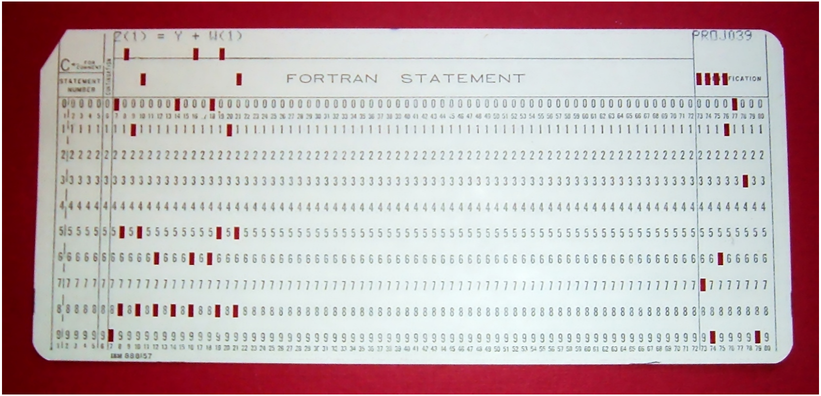

Photo by Arnold Reinhold (no copyright)

Basic Fortran Program Layout

```
program myprogram
! includes, headers, and modules
implicit none
! variable declarations
! main program
! subprograms
end program myprogram
```

The "implicit none" statement should be included in all Fortran programs. It prevents implicit typing of variables that was used in old Fortran.

Data Types

Data Types and Declarations

Intrinsic Data Types

- integer
- real
- complex
- character
- logical

Attributes to Data Types

- parameter
- kind
- size

Data Declarations

Why we use data declarations-

- data declarations help the program reference areas of memory
- standard types help create portable programs and reusable code
- codes have predictable behavior that is system independent

Fortran

Integers

- Examples: 0, 1, 23423, -3522
- No exponents, nothing after the decimal point
- Generally represented as 2,4, or 8 bytes with one bit for the sign byte

Fortran Example

implicit.f90:

```fortran
program implicit
implicit none

real :: r = 221421.23423e-7
real :: i = 3.14159e+4

print*, "r: ", r
print*, "i: ", i

end program implicit
```

Fortran Example

noimplicit.f90:

```
program noimplicit

r = 221421.23423e-7
i = 3.14159e+4

print*, "r: ", r
print*, "i: ", i

end program noimplicit
```

Fortran Example

integer_test.f90:

```fortran
program integer_test
implicit none

integer :: i
integer :: j,k, M, N=5, o
integer, parameter :: kk = 13

! integers with a precision between -10^L and 10^L
integer, parameter :: L = 5
integer, parameter :: ss = selected_int_kind(L)
integer (kind=ss) :: nn

integer (kind=ss), parameter :: mm=7

print*, "L: ", L
print*, "ss: ", ss
print*, "n: ", n
print*, "nn: ", nn
print*, "mm: ", mm
end program
```

Fortran Example

character_test.f90:

```
program character_test
implicit none
character :: a
character :: b, c, d='e', f

character (len=7):: lastname = 'Carroll'
character (len=8) :: firstname ='Hyrum'

character (len=5), parameter :: jj ='howdy'

print*, "lastname: ", lastname
print*, "firstname: ", firstname
print*, "jj: ", jj

end program character_test
```

Fortran Example

logical_test.f90:

```fortran
program logical_test
implicit none

logical :: a, b

a = .true.
b = .false.

print*, "a: ", a
print*, "b: ", b
print*, "(a .and. b): ", (a .and. b)
print*, "(a .or. b): ", (a .or. b)

end program logical_test
```

A Lesson in Semantics

IEEE 754 Standards

The IEEE 754 Floating Point standards are useful to understand for several reasons.

- they define the behavior of floating point operations in computers
- they illustrate a set of well defined machine rules, i.e. a set of language semantics
- they are essential to understanding the limits of numerical methods
- they illustrate how machines really work, and are needed to understand storage and memory issues within machines

IEEE 754 Standards

Single vs Double Precision

	sign	exponent	mantissa
single precision	1 bit	8 bits	23 bits
double precision	1 bit	10 bits	53 bits

Single Precision Numbers

- about 7 decimal digits of accuracy
- decimal exponent $< \pm 38$

```
integer, parameter :: sp = selected_real_kind(7,38)
```

Double Precision Numbers

- about 15 decimal digits of accuracy
- decimal exponent $< \pm 307$

```
integer, parameter :: dp = selected_real_kind(15,307)
```

IEEE 754 Standards

Floating Point Numbers

	sign	exponent	mantissa
single precision	1 bit	8 bits	23 bits
double precision	1 bit	10 bits	53 bits

Sign Bit

- 0 = positive
- 1 = negative

Exponent

- stored in a bias form
- bias is usually 127

Mantissa

- stored in sign magnitude form
- an implicit leading 1 is included in the mantissa. Mantissas are normally between 1 and 2 with the implied 1. For example, a mantissa of 1000 0000 0000 0000 0000 000 actually is 1.1000 0000 0000 0000 0000 000 or 1.5
- the implicit leading 1 is dropped for very small numbers, i.e. when the exponent is 0000 0000.

IEEE 754

Examples

$1.01001110101 \times 10^{110}$ (in binary)

In the IEEE standard, the bias is 127, so the exponent becomes $127 + 6 = 133$, and the leading 1 is dropped from the mantissa, so the representation becomes:

83.65625_{10}:

sign	exponent	mantissa
0	1000 0101	0100 1110 1010 0000 0000 000

IEEE 754

Examples

number	sign	exponent	mantissa
1.00000	0	0111 1111	0000 0000 0000 0000 0000 000
-1.00000	1	0111 1111	0000 0000 0000 0000 0000 000
0.00000	0	0000 0000	0000 0000 0000 0000 0000 000
2.25000	0	1000 0000	0010 0000 0000 0000 0000 000
2.75000	0	1000 0000	0110 0000 0000 0000 0000 000
-2.75000	1	1000 0000	0110 0000 0000 0000 0000 000
127.25000	0	1000 0101	1111 1101 0000 0000 0000 000
128.25000	0	1000 0110	0000 0000 1000 0000 0000 000
-5235.25000	1	1000 1011	0100 0111 0011 0100 0000 000
1.90000	0	0111 1111	1110 0110 0110 0110 0110 011
2e-38	0	0000 0001	1011 0011 1000 1111 1011 101
1.17549421e-38	0	0000 0000	1111 1111 1111 1111 1111 111
1e-38	0	0000 0000	1101 1001 1100 0111 1101 110
1e-39	0	0000 0000	0001 0101 1100 0111 0011 000
1e-41	0	0000 0000	0000 0000 0011 0111 1100 000

Floating Point Math

Special Numbers in IEEE 754

number	sign	exponent	mantissa
+0	0	0000 0000	0000 0000 0000 0000 0000 000
-0	1	0000 0000	0000 0000 0000 0000 0000 000
denormalized	0	0000 0000	nonzero
NaN	0	1111 1111	nonzero
Inf	0	1111 1111	0000 0000 0000 0000 0000 000
-Inf	1	1111 1111	0000 0000 0000 0000 0000 000

Errors in Representation

0.78000=	0	0111 1110	1000 1111 0101 1100 0010 100

$0.78 = 0.7799999713898$

0.95000=	0	0111 1110	1110 0110 0110 0110 0110 011

$0.95 = 0.9499999880791$

Fortran: Example Codes

```
program real_test
implicit none
real :: a
real :: b, c, d=5.3, e

! real with "dig" digits of precision and 10^exp in
    size
integer, parameter :: dig=6, ex=20
integer, parameter :: rpar=selected_real_kind(dig, ex)
real (kind=rpar) :: r

print*, 'dig: ', dig, 'ex: ', ex, 'rpar: ', rpar, 'r: '
    ,r

end program real_test
```

Fortran: Example Codes

```
program real_test2
implicit none
! real with "dig" digits of precision and 10^exp in
    size
integer, parameter :: dig=15, ex=3
integer, parameter :: rpar=selected_real_kind(dig, ex)
real (kind=rpar) :: r

print*, 'dig: ', dig, 'ex: ', ex, 'rpar: ', rpar, 'r: '
    ,r

! some represetations of numbers
r = 0.9d0
print*,'r (0.9d0): ',r
r = -32432.02343d+08
print*,'r (-32432.02343d+08): ',r
r = 7.434534d-23
print*,'r (7.434534d-23): ',r
r = 13124.2332d7
print*,'r (13124.2332d7): ',r

! characteristics of the variable r
```

Fortran: Example Codes

```fortran
! program real_test2 cont'd
! characteristics of the variable r
print*, 'digits: ', digits(r) ! significant digits
print*, 'epsilon: ', epsilon(r) ! least positive number that added to 1 returns a number that is > 1
print*, 'huge: ', huge(r) ! largest positive number
print*, 'maxexponent: ', maxexponent(r) ! in base 2
print*, 'minexponent: ', minexponent(r) ! in base 2
print*, 'precision: ', precision(r) ! decimal precision
print*, 'radix: ', radix(r) ! base in the model
print*, 'range: ', range(r) ! decimal exponent
print*, 'tiny: ', tiny(r) ! smallest positive number

! smaller than tiny
r = tiny(r) * tiny(r)
print*,'(tiny(r) * tiny(r)): ', r

!! bigger than huge
!r = huge(r) * 2
!print*,'r: ',r

end program real_test2
```

Fortran: Example Codes

```fortran
program binary_print
  implicit none
  character (len=32) :: bin
  integer, parameter :: rkind=selected_real_kind(5,10)
  real (kind=rkind) :: r

  print*,"Please input a number:"
  read*,r

  ! print out r as a binary number 32 digits wide
  write(bin,'(B32.32)') r
  print*,'Sign   Exponent    Mantissa'
  print*, bin(1:1),'       ',bin(2:9),'    ',bin(10:32)
  write(*,'(x,a1,6x,a8,3x,a23)') bin(1:1),bin(2:9),bin(10:32)

  write(*,'(x,a11,f30.20,a24)') "Stored as: ", r, " (displayed using write)"
  print*, "Stored as: ", r, " (displayed using print)"

end program binary_print
```

Fortran: Example Codes

```fortran
program complex_test
implicit none
complex :: a, b, d= 1.3, e =(0, -3.1)

integer, parameter :: dig=6, ex=20
integer, parameter :: rpar=selected_real_kind(dig, ex)
real (kind=rpar) :: r
complex (kind=rpar) :: c

r = (-32.02343d-02, 2.2134e+14)
c = (-32.02343d-02, 2.2134e+14)
a = r
b = c
print*, 'dig: ', dig, 'ex: ', ex, 'r: ',r
print*, 'dig: ', dig, 'ex: ', ex, 'c: ',c
print*, 'dig: ', dig, 'ex: ', ex, 'a: ',a
print*, 'dig: ', dig, 'ex: ', ex, 'b: ',b

end program complex_test
```

Loops

Fortran

Loops

Syntax 1:

```
do i = 1, 10
    ...
enddo
```

Syntax 2:

```
do i = 1, 10, 3 ! min, max, step
    ...
enddo
```

Syntax 2:

```
do while (k < 10)
    ...
    k++ ! update k
enddo
```

Fortran: Loops

```
program loop_test
implicit none
integer :: i, j, k=0

do i = 1, 10
    print*,'i: ',i
enddo
print*,'i: ',i,' (after loop)'

do j = 3, 25, 7
    print*,'j: ',j
enddo
print*,'j: ',j,' (after loop)'

do while (k < 10)
    print*,'k: ',k
    k = k + 3
enddo
print*,'k: ',k,' (after loop)'
end program loop_test
```

Write a simple Fortran program that:

1. Prints out all of the numbers between 1 and 100
2. Prints out all of the odd numbers between 0 and 100

Arrays

Arrays

Arrays :

- are an indexed list of variables
- can have one or more dimensions.
- are used to represent sets of data
- in Fortran are in contiguous blocks of memory
- indexed starting with 1 in Fortran, unless otherwise declared

Usage Examples

```
a(1) = 3.4
a(2) = 7.43e5
a(3) = -2.32e-2
...
a(3) = a(2) + 3.532
a(3) = a(2) + a(1)
```

Arrays

Declarations

- `real, dimension(10) :: a`
  - A single one-dimensional array of length 10
- `real, dimension(1000) :: b, c`
  - Two one-dimensional arrays of length 1000
- `real, dimension(-3:17) :: d`
  - A one dimensional array with a starting index of -3 and and ending index of 17
- `real, dimension(100,200) :: e`
  - A two dimensional array of size 100 x 200
- `real, dimension(50,2:41,80) :: f`
  - A three dimensional array of 50x40x80 with the second dimension starting at index 2 and ending at 41

Arrays

Why change the indexing?

- Zero indexes are commonly used in C
- Negative indexes came be used as "ghost cells" in some grids
- It simplifies programming

Fortran: Arrays

```
program arraySimpleExample
implicit none
integer :: i
integer, parameter :: n = 10
integer, dimension(n) :: results

do i = 1, n
    print *, results(i)
enddo
print*, 'results: ', results

end program arraySimpleExample
```

Fortran: Arrays

```
program array_test
implicit none
integer, dimension(10) :: i2
integer, dimension(40,40) :: i3

integer, parameter :: dig =5, ex=30
integer, parameter :: kk = selected_real_kind(dig, ex)
integer, parameter :: n = 3, m=5
real (kind=kk), dimension(n, n, m) :: jj1, jj2, jj3

integer :: i, j, k
! using loops
do i = 1, n
   do j = 1, n
      do k = 1, m
         jj1(i,j,k) = i+j+k
      enddo
   enddo
enddo
print*, 'jj1: ',jj1

! now multiply every element in the array by 7
```

Fortran: Arrays

```
! program array_test cont'd

! now multiply every element in the array by 7
do i = 1, n
   do j = 1, n
      do k = 1, m
         jj2(i,j,k) = jj1(i,j,k)* 7
      enddo
   enddo
enddo
print*, 'jj2 (with loops): ',jj2

! repeat this in array notation
jj2 = jj1 * 7
print*, 'jj2 (array notation):',jj2

end program array_test
```

Fortran: Arrays

```
program array_test2
implicit none
integer, parameter :: m=4, n=3, o=2
integer, dimension( m, n, o) :: a
integer :: i, j, k, l = 0
do i = 1, m
   do j = 1, n
      do k = 1, o
         a(i,j,k) = l
         print*,'a(',i,',',j,',',k,'):',a(i,j,k),'=',l
         l = l + 1
      enddo
   enddo
enddo
print*, 'a: ',a
print*,'a(1:3,1,1): ', a(1:3,1,1) ! low:hi[:stride]
print*,'a(2:3,1,1): ', a(2:3,1,1)
print*,'a(2:,1,1)  : ', a(2:,1,1)
print*,'a(:,1,1)   : ', a(:,1,1)
print*,'a(1,:,1)   : ', a(1,:,1)
print*,'a(1,1,:)   : ', a(1,1,:)
print*,'a(:,:,1)   : ', a(:,:,1)
```

Using Arrays

A dot product that doesn't use arrays

```fortran
program noarray
implicit none
real :: a1, a2, a3, a4, a5, a6, a7, a8, a9, a10
real :: b1, b2, b3, b4, b5, b6, b7, b8, b9, b10
real :: c

! initialize the data
call random_seed
call random_number(a1)
call random_number(a2)
call random_number(a3)
call random_number(a4)
call random_number(a5)
call random_number(a6)
call random_number(a7)
call random_number(a8)
call random_number(a9)
call random_number(a10)

call random_number(b1)
call random_number(b2)
```

Using Arrays

A dot product that doesn't use arrays (cont'd)

```
call random_number(b1)
call random_number(b2)
call random_number(b3)
call random_number(b4)
call random_number(b5)
call random_number(b6)
call random_number(b7)
call random_number(b8)
call random_number(b9)
call random_number(b10)

! calculate the dot product
c = a1*b1 + a2*b2 + a3*b3 + a4*b4 + a5*b5 + a6*b6 + &
    a7*b7 + a8*b8 + a9*b9 + a10*b10

! print out the results
print *, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10
print *, b1, b2, b3, b4, b5, b6, b7, b8, b9, b10
print*,'Results are: ',c

end program noarray
```

Using Arrays

A dot product using arrays and loops

```
program array2
implicit none
integer, parameter :: n = 10
real, dimension(n) :: a, b
integer :: i
real :: c

! initialize the data
call random_seed
do i = 1, n
    call random_number(a(i))
    call random_number(b(i))
enddo

! calculate the dot product
c = 0.0
do i = 1, n
    c = c + a(i) * b(i)
enddo

! print out the results
```

Using Arrays

A simplified dot product using intrinsic functions

```
program array3
implicit none
integer, parameter :: n = 10
real, dimension(n) :: a, b
integer :: i
real :: c

! initialize the data
call random_seed
do i = 1, n
    call random_number(a(i))
    call random_number(b(i))
enddo

! calculate the dot product
c = dot_product(a,b)

! print out the results
do i = 1, n
    print*, i, a(i), b(i)
enddo
```

Using Arrays

Simplified output using implicit looping

```
program array4
implicit none
integer, parameter :: n = 10
real, dimension(n) :: a, b
integer :: i
real :: c

! initialize the data
call random_seed
do i = 1, n
    call random_number(a(i))
    call random_number(b(i))
enddo

! calculate the dot product
c = dot_product(a,b)

! print out the results with implicit looping
print *, a
print *, b
print*,'Results are: ',c
```

Input and Output

Fortran

Reading data

The input for multiple elements can be separated by commas or by line breaks for simple numerical lists and free form input.

```
program input
implicit none
real :: a,b,i
real, dimension(3) :: c

print*,"Input a number: "
read*, i
print*, 'i: ', i
print*,"Input two numbers: "
read*, a,b
print*, 'a: ', a, 'b: ', b
print*,"Input three numbers: "
read *, c(1:3)
print*, 'c: ', c

endprogram input
```

Using Arrays

Reading data from an array

```
program array7
implicit none
integer, dimension(2,3) :: a

! Read in the data
read*, a

! print out the results
print*,'The whole array: ', a
print*,'column 1 ', a(:, 1)
print*,'column 2 ', a(:, 2)
print*,'column 3 ', a(:, 3)
print*,'row 1 ', a(1,:)
print*,'row 2 ', a(2,:)
end program array7
```

Unix and Fortran

Combining Redirection and Fortran File IO

```fortran
program array8
implicit none
integer :: i

! printout the first 6 squares
do i = 1, 6
   print*, i*i
enddo

end program array8
```

```
# make an array
gfortran array8.f90 -o array8
./array8
./array8 > array8.out
cat array8.out
```

Unix and Fortran

Combining Redirection and Fortran File IO (cont'd)

```fortran
program array7
implicit none
integer, dimension(2,3) :: a

! Read in the data
read*, a

! print out the results
print*,'The whole array: ', a
print*,'column 1 ', a(:, 1)
print*,'column 2 ', a(:, 2)
print*,'column 3 ', a(:, 3)
print*,'row 1 ', a(1,:)
print*,'row 2 ', a(2,:)
end program array7
```

```
# read in the array
gfortran array7.f90 -o array7
./array7 < array8.out
```

Unix and Fortran

Combining Redirection and Fortran File IO (cont'd II)

```
program array7
implicit none
integer, dimension(2,3) :: a

! Read in the data
read*, a

! print out the results
print*,'The whole array: ', a
print*,'column 1 ', a(:, 1)
print*,'column 2 ', a(:, 2)
print*,'column 3 ', a(:, 3)
print*,'row 1 ', a(1,:)
print*,'row 2 ', a(2,:)
end program array7
```

```
# save the output of the array7 program into array7.out
./array7 <  array8.out  > array7.out
cat array7.out
```

Unix and Fortran

Combining Redirection and Fortran File IO

```fortran
program plotdata
implicit none
integer :: i
real :: x, y
! creating a set of pairs of x, sin(x)
do i = 1, 628
   x = i / 100.0
   y = sin(x)
    print*, x,y
enddo
end program plotdata
```

```
# generate some data
gfortran plotdata.f90 -o plotdata
./plotdata
./plotdata > pdata
```

Unix and Fortran

Combining Redirection and Fortran File IO (cont'd)

Plot the data using gnuplot:

```
$ gnuplot
>plot 'pdata'
>exit
```

Write a simple Fortran program that reads in integers until it finds a number that is lower than the previous number.

Fortran So Far ...

What we've already learned:

- `implicit none`
- Intrinsic data types (integer, real, character, complex, logical)
  - Specifying the size
- Loops
- Arrays
- read*,
- print*,
- call random_seed
- call random_number(x)

Fortran Keywords

- program blocks
  - program *foo*
  - end program *foo*
- declarations
  - real :: *x*
  - integer :: *i*
  - integer, parameter :: p = selected_real_kind(15,307)
  - real (kind=p) :: *xdouble*
  - real (kind=p), dimension(20,20) :: *a*
  - real (kind=p), dimension(20,20) :: *b(0:10), c(-1:1). d(-1:1, -10:10)*

Fortran Keywords

- loops
  - do *i = 1, 30, 3*
  - do while *(k ¡ 10)*
  - enddo
- IO
  - print*, *x, i, a(3:5)*
  - read*, *i, b(4,6)*
- comments
  - ! *this is a comment*
- assignment
  - a = b + c

Intrinsic Functions

abs(x)	absolute value
cos(x)	cosine
sin(x)	sine
tan(x)	tan
acos(x)	arccos
asin(x)	arcsin
atan(x)	arctan
log(x)	natural log
exp(x)	exp
int(x)	integer part of a real value x
nint(x)	nearest integer to x
fraction(x)	fractional part of a real value x
floor(x)	greatest integer $>=$ x
real(x)	convert variable x to a real

Operators

Arithmetic Operators

- $+$: addition
- - : subtraction
- $*$: multiplication
- $/$: division (integer and real)
- $**$: exponentiation

Arithmetic Operators

Precedence

1. Sign changes
2. Exponentiations are performed
3. Multiplications and divisions
4. Additions and Subtractions

Use parenthesis to force precedence or to make the code more readable.

2+3**4 is the same as 2 + (3**4)
5*6/7+8 is the same as ((5*6)/7)+8
5*6/(7+8) is the same as (5*6)/(7+8)

Conditionals

Conditionals

if statements

if (*condition*) then

endif

Conditionals

if else statements

if (*condition*) then

else

endif

Conditionals

if statements

if (*condition*) then

else if *(condition2)* then

else if *(condition3)* then

else

endif

Conditional

Relational

```
(a == b)
(a > b)
(a >= b)
(a < b)
(a <= b)
(a != b)
```

Conditional

Logical

```
( (condition1) .and. (condition2) )

( (condition1) .or. (condition2) )

(  (condition1) .eqv. (condition2) ) ! XOR

(  (condition1) .neqv. (condition2) ) ! .not. .eqv.
```

Conditional

Logical

Precedence - highest to lowest

```
.not.
.and.
.or.
.eqv.   or .neqv.
```

Subroutines and Functions

User Defined Functions

```
type   function  fname( inputs)

end function fname
```

Subroutines

```
subroutine  sname( inputs, outputs)

end subroutine sname
```

Subroutines vs Functions

- Functions
  - Must be declared with a type and return a value (of that type)
  - Generally do NOT change the values of input data
  - Use for subprograms with simple output
- Subroutines
  - Are not required to return any value when called
  - Sometimes do affect data that is passed into the routine
  - More commonly used for subprograms that have complex output or returned values

Code Examples

- constants.f90
- otherFunctions.f90
- user_function.f90
- user_function2.f90
- user_function3.f90
- user_function4.f90
- user_function5.f90
- user_function6.f90
- user_function7.f90
- user_function8.f90

Fortran

Function and Subroutine Exercise

Write a simple Fortran program that uses both a function and a subroutine.

Fortran IO

Basic File IO

- Fortran File IO is primarily done using *read* and *write* statements
- The designation to use the standard (screen and keyboard) IO is done using *
- Output to the screen goes through unit 6
- Input from the keyboard comes from unit 5
- Other units can be opened for reading and writing to files

General Formatted Reads

```
real :: rmult

! unformatted write to standard input
print*, rmult
write(*,*) rmult

! unformatted write to unit 16
write(16,*) rmult
```

What's the second * ?

- The second * is marking the location of the format statement
- Formats are handy for making your output look pretty
- Example

  ```
  write(*, '(f15.7)') mean
  ```

  Will send the variable `mean` to the screen with a total of 15 characters with 7 after the decimal point
- Formats can get long and tricky; we'll just focus on simple ones for this class

Fortran: Formatting Output

```fortran
program binary_print
  implicit none
  character (len=32) :: bin
  integer, parameter :: rkind=selected_real_kind(5,10)
  real (kind=rkind) :: r

  print*,"Please input a number:"
  read*,r

  ! print out r as a binary number 32 digits wide
  write(bin,'(B32.32)') r
  print*,'Sign  Exponent   Mantissa'
  print*, bin(1:1),'         ',bin(2:9),'    ',bin(10:32)
  write(*,'(x,a1,6x,a8,3x,a23)') bin(1:1),bin(2:9),bin(10:32)

  write(*,'(x,a11,f30.20,a24)') "Stored as: ", r, " (displayed using write)"
  print*, "Stored as: ", r, " (displayed using print)"

end program binary_print
```

Equivalent Statements

1)

```
print*,'real multiplier'
read*, rmult
```

2)

```
write(*,*) 'real multiplier'
read(*,*) rmult
```

3)

```
write(6,*) 'real multiplier'
read(5,*) rmult
```

Fortran: Formatting Output

Format Descriptors

Syntax:

```
WRITE(unit, format, options) item1, item2, ...
READ(unit, format, options) item1, item2, ...
```

We can specify the format for both `write` and `read`:

Descriptor	Use
`I`	Integer
`F`	Real (decimal notation)
`E`	Real (scientific notation)
`A`	Character(s)
`'xyz'`	String literal
`X`	Horizontal space
`/`	Vertical space (skips lines)
`T`	TAB character

Fortran: Formatting Output

Format Descriptors

We can specify the format for both `write` and `read`:

Descriptor	Use
`rIw`	Integer
`rFw.d`	Real (decimal notation)
`rEw.d`	Real (scientific notation)
`Aw`	Character(s)
`'xyz'`	String literal
`nX`	Horizontal space
`/`	Vertical space (skips lines)
`Tc`	TAB character

w = width

r = repeat count

d = digits to display to right of the decimal

n = number of columns

c = positive integer representing column number

Other Units

Normally, we associate other units with a file name using the open statement. The basic usage is simple

```
! open the file
open(unit=27, file="newfile")

! write several values to the file
write(27,*) 33, 33.33, a, b(3), c(5:6), ' a string'

! close it
close(27)
```

Other Units

Reading simple files works the same way

```
! open the file
open(unit=27, file="oldfile")

! read several values from the file
read(27,*) 33, 33.33, a, b(3), c(5:6), str

! close it
close(27)
```

Fortran IO

- One odd thing - you can write to a unit without opening it
- A file named something like "fort.16" will be created
- This is not recommended for general use, but it is handy for debugging

Fortran

IO Status

```fortran
program ioStatus
  implicit none
  real, dimension(100) :: x, y
  integer, parameter :: channelNum = 98
  integer :: io = 0, counter = 1
  ! Use the iostat argument for the status of a IO command
  open( channelNum, file="input.dat", status='OLD', iostat=io)
  do while( io == 0)
     read( channelNum,*,iostat=io), x(counter), y(counter)
     counter = counter + 1
  end do
  close( channelNum)
  counter = counter - 2
  write(*,*),"io: ", io, "counter: ", counter
  write(*,*)," x: ", x(:counter)
  write(*,*)," y: ", y(:counter)
end program ioStatus
```

Dynamic Allocation of Arrays in Fortran

- Declare variables with undefined sizes with the "allocatable" keyword

```
real (kind=kd), dimension(:), allocatable:: a
real (kind=kd), dimension(:,:,:), allocatable :: b
integer, dimension(:,:), allocatable  :: ii
```

- Use the allocate statement to allocate memory

```
allocate(a(10000))
allocate(i(100,100)
allocate(b(50,40,300), stat=ierr)
! ierr is an integer - returns zero if successful
```

- Use the deallocate statement to free memory

```
deallocate(a)
deallocate(b,i)
```

Fortran Exercise

Exercise:

- Write a Fortran program that prompts a user for recent rainfall measurements (in inches) (example data).
- The program should display the mean.
- Before writing coding, think through the flow of the execution (draft an algorithm).
- Work in groups of two.

That's about it!

- You now have nearly all of the tools you need to write scientific code
  - There is more to learn, but you can learn this as you go
- Let's construct some code!

Fortran So Far ...

What we've already learned:

- `implicit none`
- Intrinsic data types (integer, real, character, complex, logical)
  - Specifying the size
- Loops
- Arrays
- print*, read*
- call random_seed
- call random_number(x)
- Intrinsic functions
- Arithmetic Operators (including integer / and real /)
- Conditionals (*e.g.*, if, if-else, if else if)
- Functions
- Subroutines
- IO
  - Unformatted
  - Formatted