

Goals for This Lecture:

- Understand integer arithmetic
- Understand mixed-mode arithmetic
- Understand the hierarchy of arithmetic operations
- Introduce the use of intrinsic functions

Real Arithmetic

- Valid expressions involving **real variables and constants** together with the operators **+ - * /** yield real results that you would expect.
 - **3.0+2.0 yields 5.0**
 - **3.0-2.0 yields 1.0**
 - **3.0*2.0 yields 6.0**
 - **3.0/2.0 yields 1.5**
- Division by zero is an illegal operation
 - **Generates a run-time error!**
- Negative numbers cannot be exponentiated to non-integer values
 - **Generates a run-time error!**
- Expressions involving only real numbers yield real-valued results

Integer Arithmetic

- Valid expressions involving **integer variables and constants** together with the operators **+** **-** ***** yield integer results that you would expect.
 - $3+2$ yields 5
 - $3-2$ yields 1
 - $3*2$ yields 6
- **Division of one integer by another** does not yield the answers that you would expect
- Fractional part of answer is truncated!
 - $1/2$ yields 0 (not 0.5)
 - $2/2$ yields 1
 - $3/2$ yields 1 (not 1.5)
 - $4/2$ yields 2
- **Common mistake:** forgetting that division of one integer by another yields an integer.
 - The expression $1/n$
 - will generate a zero if $n > 1$
- Also be careful of integer exponentiation!
 - $2^{**}(-3) = 1/(2^{**}3) = 1/8 = 0$

Hierarchy of Operations

- Expressions can involve multiple operations

$$y = 2.0 + 3.0 * 5.0$$

- Result depends on which order operations are evaluated $(2+3) \times 5 = 25$ or $2 + (3 \times 5) = 17$?
- In FORTRAN expressions are evaluated by the following steps in this order:
 1. Portions of expressions in parenthesis are evaluated, in order from innermost to outermost.
 2. Exponentiation is evaluated in order from right to left
 3. Multiplication and division is evaluated in order from left to right
 4. Additions and subtractions are evaluated in order from left to right

Example of expression evaluation order

- Initial expression:
 - $y = 3.0 * (2.0 + (4.0 * 2.0)) - 1.5 + 100.0 / 5.0 ** 2$
- After step 1
 - $y = 3.0 * 10.0 - 1.5 + 100.0 / 5.0 ** 2$
- After step 2
 - $y = 3.0 * 10.0 - 1.5 + 100.0 / 25.0$
- After step 3
 - $y = 30.0 - 1.5 + 4.0$
- After step 4
 - $y = 32.5$

Use of parenthesis in expressions

- Because of step 1, parenthesis can always be used to control the order of evaluation
- Parenthesis can also serve to visually clarify the order of evaluation

- Example:

both

```
y= x**3.0/4.0
```

and

```
y= (x**3.0)/4.0
```

yield the same result but in the latter case the order of evaluation is more readily apparent. This makes your program easier to read and debug

- Use of parenthesis in complex expressions constitutes good programming style!

Tips for coding complicated expressions

- Insert spaces & parenthesis into expressions to make them more readable

– Example:

```
y= y*17.0+3.14*z+x**3.0/4.0
```

or

```
y= (y*17.0) + (3.14*z) + ((x**3.0)/4.0)
```

the latter is more readable!

– Example:

```
y=y*17.0+3.1*z*(1.22+15.4*(z+2.0)**3)+x**3.0/4.0
```

or

```
y1 = y*17.0
```

```
y2 = 3.1*z*(1.22+15.4*(z+2.0)**3)
```

```
y3 = x**3.0/4.0
```

```
y = y1+y2+y3
```

the latter is more readable and much easier to debug!

Mixed-mode Arithmetic

- Arithmetic operations can be applied to combinations of integers and real variables (or constants or expressions)
- These are called mixed-mode operations
- How do these evaluate?
 - $3/2$ yields 1 (integer result)
 - $3.0/2.0$ yields 1.5 (real result)
 - $3.0/2$ yields 1.5 (real result)
 - $3/2.0$ yields 1.5 (real result)
- Rule for mixed-mode arithmetic:
 - *FORTRAN automatically converts the integer to a real number and evaluates the expression according to real arithmetic rules*
- Automatic mode conversion does not occur until a real value and an integer value occur in the same operation
- This means that portions of an expression can evaluate to integer values while other portions evaluate to real values
- Example:
 - Initial expression: $1/3 + 1.0/2$
 - Evaluation step 1: $0 + 0.5$
 - Evaluation step 2: 0.5

Mixed-mode Exponentiation

- In general, avoidance of mixed-mode operations are desirable. The exception to this is exponentiation.
- Exponentiating a real value
 - If one exponentiates a real value to an integer power n the compiler treats this as shorthand for “multiply the real value by itself n times”
 $2.0^{**}5 = 2.0 * 2.0 * 2.0 * 2.0 * 2.0$
 $-3.0^{**}3 = (-3.0) * (-3.0) * (-3.0)$
 - If one exponentiates a real value to a real power the result is evaluated via logarithms using the formula $y^x = e^{x \ln(y)}$
 $2.0^{**}5.0 = \text{exp}(2.0 * \text{log}(y))$
 - It is not possible to exponentiate a negative real value to a real power
 $(-2.0)^{**}2.1$ yields a NaN (Not a Number)
- Exponentiating an integer value
 - Exponentiating an integer value to a real power is done with logarithms as with exponentiating a real value to a real power
 - Exponentiating an integer to an integer power works just like exponentiating a real value to an integer power
 - Lesson: Exponentiate to integer powers when possible
 - $x^{**}2$ is faster and more accurate than $x^{**}2.0$

Assignment Statements

- Assignment statements can be used to cast real values into integer variables and vice versa.
- The following two rules apply:
 - Integer values will cast into real variables by converting the integer to a real of the same numerical value.
 - Example (assume x is a real variable):
 - `x=1` assigns value of 1.0 to x
 - `x=-2` assigns value of -2.0 to x
 - `x=1/2` assigns value of 0.0 to x (1/2 still evaluates to 0)
 - Real values will cast into integer variables by truncating the fractional part of the real value.
 - Example (assume i is an integer variable):
 - `i=1.0` assigns value of 1 to i
 - `i=1.1` assigns value of 1 to i
 - `i=0.999` assigns value of 0 to i

Intrinsic Functions

- FORTRAN has a built in capability to evaluate many common mathematical and algorithmic functions.
- Most languages have some capability to do this but in the case of FORTRAN these functions are standardized.
 - This means that they do not vary from compiler to compiler
 - Not true for other languages
- These built in functions are called intrinsic functions
- There are a wide variety of functions that operate on all fortran data types
- These include trigonometric functions, logarithms, square roots, absolute values, exponentials, etc.

Intrinsic Function Usage

- Intrinsic functions can be utilized anywhere a value is required in an expression
- Usage:
`function_name(input1,input2,...)`
- The inputs to a function are called arguments
- A functions can yield real, integer, or other values depending on it's particular capability
- A list of intrinsic functions can be found in table 2-6 or Appendix B of your textbook
- The list of functions will describe the required arguments

Some Really Useful Intrinsic Functions

Function	Argument Type	Result Type	Comment
<code>sqrt(x)</code>	real	real	Square root
<code>abs(x)</code> or <code>abs(i)</code>	real integer	real integer	Absolute value
<code>sin(x)</code>	real	real	Sine
<code>cos(x)</code>	real	real	Cosine
<code>tan(x)</code>	real	real	Tangent
<code>exp(x)</code>	real	real	Exponential
<code>log(x)</code>	real	real	Natural logarithm
<code>log10(x)</code>	real	real	Base-10 logarithm
<code>asin(x)</code>	real	real	Inverse sine
<code>acos(x)</code>	real	real	Inverse cosine
<code>atan(x)</code>	real	real	Inverse tangent
<code>max(a,b)</code>	real/integer	real/integer	Maximum of A & B
<code>min(a,b)</code>	real/integer	real/integer	Minimum of A & B
<code>nint(x)</code>	real	integer	Nearest integer

Example 1 of Intrinsic Function Use

```
! Purpose: Demonstrate intrinsic function usage
! Author:  F. Douglas Swesty
! Date:    9/12/2005
program demo_intrinsics
implicit none    ! Turn off implicit typing
real :: theta    ! Input angle
real :: sine_theta, cos_theta, tan_theta    ! Variables to hold trig values

                                ! Prompt the user for the angle
write(*,*) "Enter angle (in radians):"
read(*,*) theta                ! Read in the angle

sine_theta = sin(theta)        ! Calculate sine
cos_theta = cos(theta)         ! Calculate cosine
tan_theta = tan(theta)         ! Calculate tangent
                                ! Output the results
write(*,*) " Sine, Cosine, and Tangent are: "
write(*,*) sine_theta,cos_theta,tan_theta
stop                            ! Stop execution of the program
end program demo_intrinsics
```

Example 2 of Intrinsic Function Use

```
! Purpose: Demonstrate max, min, & abs intrinsic function usage
! Author:  F. Douglas Swesty
! Date:    9/12/2005
program demo_max_min
implicit none    ! Turn off implicit typing
integer :: i1, i2    ! Input integers
integer :: max_i    ! Variable to hold max value
integer :: min_i    ! Variable to hold min values
integer :: abs_max_i ! Variable to hold absolute value of maximum
integer :: abs_min_i ! Variable to hold absolute value of minimum
write(*,*) "Enter i1 & i2:"    ! Prompt the user for integers
read(*,*) i1,i2                ! Read in the integers
max_i = max(i1,i2)              ! Calculate maximum of two arguments
min_i = min(i1,i2)              ! Calculate minimum of two arguments
abs_max_i = abs(max_i)          ! Calculate absolute value of maximum
abs_min_i = abs(min_i)          ! Calculate absolute value of minimum
                                ! Output the results
write(*,*) " max = ",max_i
write(*,*) " min = ",min_i
write(*,*) " Absolute value of max = ",abs_max_i
write(*,*) " Absolute value of min = ",abs_min_i
stop                            ! Stop execution of the program
end program demo_max_min
```

Assignments

- Read Sections 2.4-2.6, 2.10 of Chapman (FORTRAN 90/95)