

# Goals for This Lecture:

- Understand integer, real, and character constants & variables
- Introduce the use of declaration statements
- Understand the parameter attribute
- Introduce the assignment statement
- Introduce list directed write & read statements

# Constants & Variables

- FORTRAN Constant: A data object that is defined before the program is executed and whose value never changes
- FORTRAN Variable: A data object that can change it's value during the execution of the program
- Five intrinsic, i.e. built-in, types of variables & constants in FORTRAN:
  - integer
  - real
  - complex
  - logical
  - character

# Integer Constants

- An integer constant is any number that does not contain a decimal point
- It can be written with either plus or minus signs in front of it.
- No commas are allowed within the constant
- Valid integer constants:
  - 0
  - 347
  - +12543
  - 53
- Examples of illegal attempts to form constants
  - 9,999 (no commas allowed)
  - 23.0 (no decimal points allowed; this is regarded as a REAL constant)

# Real Constants

- A real, or **floating-point**, constant is a number that contains a decimal point and an optional integer exponent
- Real constants can be written with or without an exponent

3.14            (3.14)

3.14e0        (3.14)

314.0e-2      (3.14)

5.25e-13     (5.25 x 10<sup>-13</sup>)

-125.0e+23   (-1.25 x 10<sup>25</sup>)

Invalid examples:

1,000.0      (no commas allowed)

123e2        (a decimal point is required in the mantissa if the exponent is specified)

3.14e0.0     (exponent must be an integer)

- Exponents can also be specified with additional precision using a “d” instead of an “e”

3.14d0

This is called a double-precision constant

# Complex Constants

- A complex constant is a pair of floating-point numbers specified in the form  
*( real\_part , imaginary\_part )*
- Both the real and imaginary part should be of the same type, i.e. real of double-precision real
- Valid examples:  

(3.14,0.0)	(3.14+0i)
(1.25,-2.2)	(1.25-2.2i)
(1.23e10,1.6e-13)	(1.23x10 <sup>10</sup> + 1.6x10 <sup>-13</sup> i)
(5.25d-13,1.0d0)	(5.25 x 10 <sup>-13</sup> +1.0i)
- Invalid examples:  

(1,000.0,2.2)	(no extra commas allowed)
(123e2,1.0	(invalid real component)

# Character Constants

- A character constant is a string of alphanumeric characters enclosed in a set a single (') or double (") quotes
- The minimum number of characters in a constant is 1
- The characters in a constant can be any keyboard character, not just the ones legal for FORTRAN programs
- If an apostrophe is needed in the quote it can be written using double quotes
  - "Don't think this is illegal!"
  - Or by using two quotes in a row
  - 'Don''t think this is illegal'
- Valid examples:
  - "This is a constant"
  - " " (blank spaces are OK)
  - 'My name is Doug'
- Invalid examples:
  - My name is Doug (no quotes)
  - 'My name is Doug" (mismatched quotes)
  - ' 'My name is Doug' (unbalanced quotes)
- Character constants are often used to output descriptive information in a write statement
  - write(\*,\*) " Velocity = ",vel

# Logical Constants

- A logical constant can only take on two values which are specified by `.TRUE.` and `.FALSE.`
- Valid examples:
  - `.true.`
  - `.TRUE.`
  - `.false.`
  - `.FALSE.`
- Invalid examples:
  - `.TRUE` (Unbalanced periods)
  - `FALSE` (no periods; will be treated as a variable named "FALSE")
- Logical constants are usually used in logical expressions that control program execution

# Parameters

- Constants can be assigned to a variable through the use of the **parameter attribute** the declaration statement
- Example:  
`real, parameter :: pi=3.1415923`
- The parameter pi can now be used in any place where a real variable, constant, or expression would be utilized
- Parameters are a great way to make sure that constants are consistent throughout your program

# Assignment Statements

- Variables in FORTRAN can be assigned values by means of an *assignment statement*
- Form:  
`variable = expression`
- The “=” is referred to as the assignment operator
- It does not indicate equality
- Statements like  
`x = x + 1.0`  
are perfectly valid.
- This statement says “take the value of x, add one to it, and assign the new value back to x”

# Arithmetic Operators

- The following operators can be utilized in numerical expressions in FORTRAN:
  - + addition
  - subtraction
  - \* multiplication
  - / division
  - \*\* exponentiation
- These are all binary operators
  - They involve two operands
- The + and – operators can also occur as unary operators
- Examples:
  - +3.14
  - velocity

# Arithmetic Operator Rules

- No two operators may occur sequentially (one after the other)

$a*-b$  Illegal

$a*(-b)$  Legal

$a**-2$  Illegal

$a**(-2)$  Legal

- Negative values can not be exponentiated to a non-integer power or a negative power

$(-1.0)**2$  Legal

$(-1.0)**2.0$  Illegal, Compiler may allow this to work but it is a very bad idea!

$(-1.0)**2.1$  Illegal; gives a NaN (Not A Number)

$(3.14)**0.5$  Legal

- Division by zero (either integer, real, or complex) is not permissible

$1.0/0.0$  Illegal

$2/0$  Illegal

# Arithmetic Expressions

- No two operators may occur sequentially (one after the other)
  - Illegal:  $a*-b$
  - Legal:  $a*(-b)$
  - Illegal:  $a**-2$
  - Legal:  $a**(-2)$
- Parenthesis can be used to control the order of operations
  - Example:
    - $2**5-2 = 32-2 = 30$
    - $2**(5-2) = 2**3 = 8$
- **Important:** all variables must be assigned values before they can be utilized in an expression

# Using Arithmetic Expressions

```
! Purpose: Convert Fahrenheit to Celsius
! Author:  F. Douglas Swesty
! Date:    9/9/2005
program conv_f_to_c

implicit none      ! Turn off implicit typing
real :: temp_f    ! Temperature in Fahrenheit
real :: temp_c    ! Temperature in Celsius

temp_f = 85       ! Set temperature in Fahrenheit

temp_c = (temp_f-32.0)*5.0/9.0 ! Convert temperature from
                                ! degrees Fahrenheit to degrees
                                ! Celsius

write(*,*) " Temperature in degrees Celsius = ",temp_c

stop              ! Stop execution of the program

end program conv_f_to_c
```

# List directed Write & Read statements

- It would be nice to have a way of getting data into a and out of a program interactively.
- An easy way to do this are list-directed READ and WRITE statements.
- List-directed means that the types of variables, expressions, or constants in the list determine the format of the data
- Example:  
`write(*,*) "k = ",k`
- This statement outputs the two items in the list:
  - The character constant `"k = "`
  - The integer variable `k`

# List directed READ statements

- Statement takes the form:  
`read(*,*) var1,var2,var2,...`
- The list can have as many variables as you would like.
- The input data is read from your keyboard.
- Each item in the input list must be separated by a comma, blank space, or on it's own line.
- The format of each data item entered must match the variable type.
- The number of data items entered must be greater than or equal to the number of variables.

# List directed WRITE statements

- Statement takes the form:

```
write(*,*) item1,item2,item3,...
```

- The list can consist of variables, constants, or expressions
- The list can have as many items as you would like
- The data is output to your screen
- The format of each data item output will match the item type

# Example of List Directed WRITE/READ Use

```
! Purpose: Fahrenheit to Celsius Calculator
! Author:  F. Douglas Swesty
! Date:    9/9/2005
program f_to_c_calc
implicit none    ! Turn off implicit typing
real :: temp_f  ! Temperature in Fahrenheit
real :: temp_c  ! Temperature in Celsius

                                ! Prompt the user for the temperature
write(*,*) "Enter Temperature (in Fahrenheit):"
read(*,*) temp_f                ! Read in the temperature

temp_c = (temp_f-32.0)*5.0/9.0 ! Convert temperature from
                                ! degrees Fahrenheit to degrees
                                ! Celsius

write(*,*) "Temperature in degrees Celsius = ",temp_c

stop                            ! Stop execution of the program
end program f_to_c_calc
```

# Reading Assignments

- Read Sections 2.7-2.9, 2.11-2.12 of Chapman (FORTRAN 90/95)