Goals for This Lecture:

• Introduce iterative (or counting) DO loops
• Understand the rules of iterative DO loop execution
• Understand how to use DO loops to accomplish summation
• Understand nesting of DO loops
The iterative DO loop construct

- The iterative DO loop construct executes the statement block in the body of the loop a specified number of times

**Form:**
```
do ivar = istart, iend, incr
  statement block1
endo
```

**Example of use:**
```
do i=1,10,2
  write(*,*) ' i = ',i
endo
```

- The integer quantities `istart`, `iend`, and `incr` are known as the **parameters** of the do loop
- The integer variable `ivar` is known as the DO loop index
Iterative DO loop behavior

- The parameters istart, iend, and incr may be integer variables, parameters, functions, or expressions.
- Steps in execution of a iterative DO loop:
  1. At the beginning of DO loop execution the index is assigned the value istart. If $ivar \times incr \leq iend \times incr$ the loop executes the statement block in the body of the loop.
  2. After the statements in the body of the loop have been executed the index is incremented by the value incr ($ivar = ivar + incr$)
  3. If the $index \times incr \leq iend \times incr$ the loop executes again
  4. Step 2 is repeated as long $index \times incr \leq iend \times incr$
- The number of iterations executed by the DO loop is given by
  $$iter = \frac{iend - istart + incr}{incr}$$
- The increment parameter $incr$ is optional. If omitted the value of the increment defaults to unity.
Execution flow for iterative DO loop construct

- **IMPORTANT:** The DO-loop index cannot be reassigned anywhere in the body of the loop
- This means that the index cannot appear on the left-hand-side of an assignment statement within the body of the loop
Application: Calculate N!

- \( N! = N \times (N-1) \times (N-2) \times \ldots \times 1 \)
- \( 0! = 1 \)

- **Initial Algorithm:**
  1. Start
  2. Prompt user for \( N > 0 \)
  3. If \( N == 0 \) set factorial = 1
  4. If \( N == 1 \) set factorial = 1
  5. If \( N >= 2 \)
  6. DO from \( i = 2, \ldots, N \)
  7. multiply nfact by \( i \)
  8. ENDDO
  9. Endif
  10. Write out value of factorial
  11. Stop
Application: Calculate N!

- \( N! = N \times (N-1) \times (N-2) \times \ldots \times 1 \)
- \( 0! = 1 \)

- Refined Algorithm (makes use of DO loop behavior when index*incr > iend*incr):
  1. Start
  2. Prompt user for N > 0
  3. Set factorial = 1
  4. DO from i = 2,\ldots,N
  5. multiply nfact by i
  6. ENDDO
  7. Write out value of factorial
  8. Stop
Example: Factorial Calculator

```fortran
! Purpose: Calculate a factorial
! Author: F. Douglas Swesty
! Date: 9/26/2005
program factorial
implicit none ! Turn off implicit typing
integer :: n    ! Variable to hold value of N
integer :: i    ! Loop index
integer :: nfact ! Variable to hold N-factorial
write(*,*) "Enter N :"  ! Prompt the user for N
read(*,*)  n               ! Read in N

nfact = 1                           ! Initialize nfact to unity
do i = 2,n,1         ! Initiate loop
   nfact = nfact*i                 ! Multiply by next factor
endo             ! Terminate loop
write(*,*) ' N! = ',nfact
stop                                    ! Stop execution of the program
end program factorial
```
N! a different way:

- Decrementing Algorithm (makes use of DO loop behavior with incr < 0):
  1. Start
  2. Prompt user for N > 0
  3. If( N <= 1) then
  4. set factorial = 1
  5. write out value of factorial
  6. stop
  7. endif
  8. Set factorial = N
  9. DO from i = N,N-1,…,1
  10. multiply nfact by i
  11. ENDDO
  12. Write out value of factorial
  13. Stop
Example: Factorial Calculator

```fortran
! Purpose: Calculate a factorial in hi-to-low order
! Author: F. Douglas Swesty
! Date: 9/26/2005

program factorial2
implicit none  ! Turn off implicit typing
integer :: n          ! Variable to hold value of N
integer :: i           ! Loop index
integer :: nfact      ! Variable to hold N-factorial
write(*,*) "Enter N:"
read(*,*)  n    ! Prompt the user for N
if(n <=1 ) then
   write(*,*) ' N! = 1'
   stop
endif

nfact = n            ! Initialize nfact to unity
n = n,1,-1    ! Initiate loop
nfact = nfact*i    ! Multiply by next factor
endo             ! Terminate loop

write(*,*) ' N! = ',nfact    ! Stop execution of the program
stop
end program factorial2
```
A Major Application technique: Using DO loops for summation

Suppose we want to calculate summation of a sequence of the form:

\[ \gamma = \sum_{n=1}^{N} \frac{1}{n^3} \]

This can be easily accomplished with iterative DO loops

Algorithm:
1. Start
2. Prompt user for N
3. Read in N
4. Initialize sum to zero
5. DO for i=1,2,…,N
6. add 1/(i^3) to sum
7. ENDDO
8. Write out sum
9. Stop
Summation Code

! Purpose:  Sum the sequence 1/(n**3) from 1 to N
! Author:   F. Douglas Swesty
! Date:     9/26/2005
program summation
implicit none          ! Turn off implicit typing
real :: sum          ! Variable to hold sum
integer :: n        ! Variable to hold length of sequence
integer :: i        ! Loop index
write(*,*) ' Enter N >= 1:' ! Prompt user for N
read(*,*) n        ! Read in N
sum = 0.0           ! Initialize sum
do i=1,n,1            ! Initiate loop
   sum = sum+1.0/((1.0*i)**3)  ! Add next term to sum
enddo     ! Terminate loop
write(*,*) " sum = ",sum
stop            ! Stop execution of the program
end program summation
Nesting of DO loops

• DO loops can be nested within one another
• If both loops are iterative loops they must have different index variables

Example of nested loops:

```fortran
    do j=1,10,2
        do i=1,10,2
            write(*,*) ' i,j = ',i,j
        enddo
    enddo
```

Example two:

```fortran
    x = 0.0
    do while (x < 35.0)
        do i=1,10,2
            write(*,*) ' i,j = ',i,j
        enddo
        x = x+1.0
    enddo
```
Example: Factorial Calculator

! Purpose: Repeating factorial calculator
! Author: F. Douglas Swesty
! Date: 9/26/2005

program factcalc
implicit none ! Turn off implicit typing
integer :: n ! Variable to hold value of N
integer :: i ! Loop index
integer :: nfact ! Variable to hold N-factorial

n = 1
do while(n >= 1) ! While the import value is valid
    write(*,*) "Enter N :" ! Prompt the user for N
    read(*,*) n ! Read in N

    nfact = 1 ! Initialize nfact to unity
    do i = 2,n,1 ! Initiate loop
        nfact = nfact*i ! Multiply by next factor
    enddo ! Terminate loop
    write(*,*) ' N! = ',nfact

enddo ! Stop execution of the program
end program factcalc
Reading Assignment

– Read Sections 4.1, 4.3