

Goals for This Lecture:

- Introduce Some File I/O Basics
- Learn the basic usage of the OPEN & CLOSE statements
- Learn how to use the iostat= clause in the READ statement

ASCII File Structure

- Files consist of many lines of data
- In FORTRAN each line of the file is called a record
- FORTRAN can read or write ASCII files one record at a time
- (FORTRAN 2003 also has stream based I/O capabilities like C++)
- When we are accessing one record after another we are accessing the file via *sequential access*
- When we access a record and then access another non-adjacent record we are accessing the file via *direct access*
- FORTRAN can support both types of access but ordinary ASCII files are by default accessed as sequential files

Logical Units Numbers in FORTRAN I/O

- FORTRAN has a mechanism to read from and write to files
- This mechanism relies on the concept of a Logical Unit Number (LUN) which is attached to a specified file
- Data are written to, and read from, logical units via write & read statements
- Example:

```
write(17,*) "Hello World!"
```

“Hello World!” character data is written to logical unit number 17

Assigning Logical Unit Numbers to Files

- LUNs are assigned to files with the **open statement**
- LUNs are released from files with the **close statement**
- LUNs in the range of 1-99 are available for assignment to a file
- Certain LUNs are usually reserved (not standard but widely implemented):
 - Usually LUN 5 is connected to STDIN
 - Usually LUN 6 is connected to STDOUT
- Good idea to avoid use of LUNs 1-10
- There may be an upper limit to how many LUNs can be attached to files by any one program
- Each LUN can only be attached to one file at a time

Assigning Logical Unit Numbers to Files

- LUNs are assigned to files with the **open statement**
- LUNs are released from files with the **close statement**
- LUNs in the range of 1-99 are available for assignment to a file
- Certain LUNs are usually reserved (not standard but widely implemented):
 - Usually LUN 5 is connected to STDIN
 - Usually LUN 6 is connected to STDOUT
- Good idea to avoid use of LUNs 1-10
- There may be an upper limit to how many LUNs can be attached to files by any one program
- Each LUN can only be attached to one file at a time

Example: Writing data to a file

```
! Purpose: Show how to write data to a file
! Author:  F. Douglas Swesty
! Date:    10/14/2005
program dwrite1
implicit none      ! Turn off implicit typing
real :: x =1.0, y=2.0, z=3.14 ! Define some fake data

open(unit=11,file='output.dat',status='REPLACE') ! Open
the file
write(11,*) "x, y, and z values are" ! Write out a header
write(11,*) " x = ",x                ! Write out data
write(11,*) " y = ",y
write(11,*) " z = ",z
close(unit=11)                        ! Close the file
stop                                  ! Halt execution
end program dwrite1
```

Good Programming Tip

- Use an integer parameter or integer variable for an LUN instead of hard-coding in the integer.
- If you later need to change the LUN number it only needs to be changed in the parameter or declaration statement

Example: Writing data to a file

```
! Purpose: Show how to use a integer parameter LUN
! Author: F. Douglas Swesty
! Date: 10/14/2005
program dwrite2
implicit none      ! Turn off implicit typing
integer, parameter :: lun1=11 ! Define an LUN for the output file
real :: x =1.0, y=2.0, z=3.14 ! Define some fake data

open(unit=lun1,file='output.dat',status='REPLACE') ! Open the file
write(lun1,*) "x, y, and z values are" ! Write out a header
write(lun1,*) " x = ",x                ! Write out data
write(lun1,*) " y = ",y
write(lun1,*) " z = ",z
close(unit=lun1)                       ! Close the file
stop                                    ! Halt execution
end program dwrite2
```

The OPEN statement

- FORM: `open(open_list)`
- *open_list* is a series of clauses specifying information about how the file should be treated during I/O. There are many possible clauses.
- Five most important clauses:
 1. `UNIT=lun#` clause. Specifies what LUN is to be assigned to the file.
 2. `FILE=fname` clause. Specifies the name of the file.
 3. `STATUS=fstat` clause. Specifies the file status. *fstat* is one of: `'OLD'`, `'NEW'`, `'REPLACE'`, `'SCRATCH'`, or `'UNKNOWN'`
 4. `ACTION=actstr` clause. Specifies whether a file is to be opened for writing, reading, or both. *actstr* is one of `'READ'`, `'WRITE'`, or `'READWRITE'`
 5. `IOSTAT=ierror` clause. *ierror* is an integer variable containing the error status of the open statement (zero indicates no errors)
- Clauses 1. & 2. are mandatory. The others are optional.

The **STATUS=** Clause

- This clause is optional
- Behavior:
 - If **STATUS='OLD'** is specified the file must already exist or the `iostat` variable will return a non-zero error code.
 - If **STATUS='NEW'** is specified the file **must not** exist or the `iostat` variable will return a non-zero error code. The file will be created.
 - If **STATUS='REPLACE'** is specified the file is overwritten if it exists or is created if it does not exist.
 - If **STATUS='SCRATCH'** is specified the file is created but is deleted after the file is closed.
 - If **STATUS='UNKNOWN'** is specified the status of the file is system dependent.
 - If the **STATUS=** clause is omitted the default status is **'UNKNOWN'**

The **ACTION=** Clause

- This clause is optional
- Behavior:
 - If **ACTION= 'READ'** is specified the file can only be read from.
 - If **ACTION= 'WRITE'** is specified the file can only be written to.
 - If **ACTION= 'READWRITE'** is specified the file can be both written to and read from.
 - If **ACTION=** clause is omitted the default value is **'READWRITE'**

The **IOSTAT=** Clause

- This clause is optional
- If **IOSTAT=ierror** is specified in the open statement the integer variable `ierror` contains an error code to indicate that the open statement was successful (if `ierror` is zero) or a failure (`ierror` has non-zero value)
- The **IOSTAT=ierror** clauses can also be added to the READ statement to check for errors encountered during a READ operation

- For example

```
read(11,*,iostat=ierror) xdata, ydata
if(ierror /= 0) then
  write(*,*) ' Read statement failed'
  stop
endif
```

The CLOSE statement

- FORM: `close(open_list)`
- *open_list* is a series of clauses specifying information about how the file should be treated during closure. There are many possible clauses.
- The most important clause:
 - `UNIT=lun#` clause. Specifies what LUN is to be assigned to the file.
 - This clause is mandatory.

Example: Reading data from a file

```
! Purpose: Show how to read data from a file
! Author:  F. Douglas Swesty
! Date:    10/14/2005
program dread1
implicit none      ! Turn off implicit typing
real :: x, y, z    ! Define some variables to hold data
integer, parameter :: lun1=11 ! Define an LUN for the input file
integer :: ierror  ! I/O status variable
open(unit=lun1,file='input.dat',status='OLD',IOSTAT=ierror) ! Open the file
if(ierror /= 0) then ! Check the I/O status
  write(*,*) 'Data file open failed'
  stop
endif
read(lun1,*) x      ! Read in x
read(lun1,*) y      ! Read in y
read(lun1,*) z      ! Read in z
close(unit=lun1)    ! Close the file
stop                ! Halt execution
end program dread1
```

Example: Reading in an unknown amount of data

```
! Purpose: Calculate average of data from a file
! Author:  F. Douglas Swesty
! Date:    10/14/2005
program average
implicit none      ! Turn off implicit typing
real :: x, sum, average  ! Define needed variables
integer :: ndata        ! Number of data values
integer, parameter :: lun1=11 ! Define an LUN for the input file
integer :: ierror      ! I/O status variable
open(unit=lun1,file='input.dat',status='OLD',IOSTAT=ierror) ! Open the file
if(ierror /= 0) then   ! Check the I/O status
  write(*,*) 'Data file open failed'
  stop
endif
sum = 0.0              ! Initialize sum variable
ndata = 0              ! Initialize data counter
do
  read(lun1,*,iostat=ierror) x          ! Read in x
  if(ierror == 0) then
    sum = sum+x
  else
    write(*,*) 'Read in ',ndata, ' data points'
    close(unit=lun1)
  endif
enddo
avg = sum/float(ndata) ! Calculate average
write(*,*) 'average = ',avg ! Output the average

stop                ! Halt execution
end program average
```

File Positioning

- Occasionally you may want to backup within a file and write or read a line (record) a second time.
- FORTRAN provides a way to manage positions within a file with the **backup** and **rewind** statements
- The **backup** statement moves backward one record (one line) in the file each time it is executed
 - FORM: **backup**(*lun*)
- The **rewind** statement moves back to the beginning of the file (first record a.k.a. first line) each time it is executed
 - FORM: **rewind**(*lun*)

Reading Assignment

- Read Sections 6.1-6.3