OPENING FILES

Opening files

In Fortran, files are designated by unit numbers.
Values from 0 to 6 are reserved for standard I/O.

```
OPEN(unit=10,file=‘data.txt’,action=‘read’)
```

```
OPEN(unit=11,file=‘results.txt’,action=‘write’)
! Action=‘readwrite’ is also possible.
```

Closing files

```
CLOSE(unit=10)
```

```
OPEN(unit=11)
```

READ AND WRITE

Input and Output Statements

- Basic instructions:
  - READ – reads input from a standard input device or a specified device or file.
  - WRITE – writes data to a standard output device (screen) or to a specified device or file.
  - FORMAT – defines the input or output format.

READ Statement

- Format controlled READ:
  - Syntax: **READ(dev_no, format_label) variable_list**
  - Read a record from dev_no using format_label and assign results to variables in variable_list
  - Ex:

```
       READ(5,1000) A,B,C
  1000 FORMAT()
```

  - Ex:

```
        READ(5,"(3F12.4)") A,B,C
```

  - Device numbers 1-7 are defined as standard I/O devices and 1 is the keyboard, but 5 is also commonly taken as the keyboard (used to be card reader)
  - Variable_list can include implied DO such as:

```
READ(5,"(10F12.4)")(A(I),I=1,10)
```

WRITE Statement:

- Format controlled WRITE
  - Syntax: **WRITE(dev_no, format_label) variable_list**
  - Write variables in *variable_list* to output *dev_no* using format specified in format statement with *format_label*

Ex: write(*,*) A,B,Key

Ex:
```
WRITE(6,1000) A,B,KEY
 1000 FORMAT(F12.4,E14.5,I6)
```

Ex:
```
WRITE(6,"(F12.4,E14.5,I6)") A,B,KEY
```

```
Output:
|----+----o----+----o----+----o----+----|
    1234.5678  -0.12345E+02    12
```

  - Device number 6 is commonly the printer but can also be the screen (standard screen is 2)
  - Each WRITE produces one or more output lines as needed to write out variable_list using format statement.
  - Variable_list can include implied DO such as:

```
WRITE(6,2000)(A(I),I=1,10)
```

```
program MD
!=========================================================================
! This program implements a simple molecular dynamics simulation,
!   using the velocity Verlet time integration scheme. The particles
!   interact with a central pair potential.
!
! Author:   Bill Magro, Kuck and Associates, Inc. (KAI), 1998
!=========================================================================
  implicit none

  ! simulation parameters:
  ! ndim = dimensionality of the physical space
  ! nparts = number of particles
  ! nsteps = number of time steps in the simulation
  ! mass = mass of the particles
  ! dt = time step
  integer, parameter :: ndim=3, nparts=500, nsteps=1000
  real(8) :: mass=1.0d0
  real(8) :: dt=1.0e-4

  ! simulation variables
  real(8) :: box(ndim)  ! dimensions of the simulation box
  real(8) :: position(ndim,nparts), velocity(ndim,nparts)
  real(8) :: force(ndim,nparts), accel(ndim,nparts)
  real(8) :: potential, kinetic, E0, xx, rij(ndim),d,v,dv
```

TYPES

Integers

Typically ±2147483647 (-2^{31} to $2^{31}-1$)

INTEGER, INTEGER(4)

Sometimes $\pm 9.23 \times 10^{17}$ (-2^{63} to $2^{63}-1$)

INTEGER(8)

Integers

Fortran uses integers for:

- Loop counts and loop limits
- An index into an array or a position in a list
- An index of a character in a string
- As error codes, type categories etc.

Also use them for purely integral values

E.g. calculations involving counts (or money)

They can even be used for bit masks (see later)

Reals

- Reals are held as floating-point values

These also have a finite range and precision

It is essential to use floating-point appropriately

Real Constants

• Real constants must contain a decimal point or an exponent

They can have an optional sign, just like integers

The basic fixed-point form is anything like:

123.456, -123.0, +0.0123, 123., .0123

0012.3, 0.0, 000., .000

• Optionally followed E or D and an exponent

1.0E6, 123.0D-3, .0123e+5, 123.d+06, .0e0

1E6 and 1D6 are also valid Fortran real constants

Complex Numbers

This course will generally ignore them

If you don’t know what they are, don’t worry

These are (real, imaginary) pairs of REALs

i.e. Cartesian notation

Constants are pairs of reals in parentheses

E.g. (1.23, -4.56) is 1.23-4.56i

or (-1.0e-3, 0.987) is -0.001+0.987i

Exercise: Which of these are legal Fortran constants? What are their types?

(i) .	(ii) 3.	(iii) 3.1
(iv) 31	(V) 0.	(vi) +2
(vii) –E18	(viii) “ACHAR(61)”	(ix) 3 500
(x) 4,800,000	(xi) “X or Y”	(xii) “X”//“Y”
(xiii) 4.8E6	(xiv) 5000E–3	(xv) “VAT 69”
(xvi) 6.6_big	(xvii) (1, –1)	(xviii) 007
(xix) 1E	(xx) –630958813365	

Using KIND

Declaring variables etc. is easy

```
INTEGER, PARAMETER :: DP = &
SELECTED_REAL_KIND(12)
REAL(KIND=DP) :: a, b, c
REAL(KIND=DP), DIMENSION(10) :: x, y, z
```

Using constants is more tedious, but easy

0.0_DP, 7.0_DP, 0.25_DP, 1.23_DP, 1.23E12_DP,

0.1_DP, 1.0E-1_DP, 3.141592653589793_DP

That's really all you need to know . . .

DOUBLE PRECISION

You can use it just like REAL in declarations

Using KIND is more modern and compact

REAL(KIND=KIND(0.0D0)) :: a, b, c

Constants use D for the exponent – 1.23D12 or 0.0D0

REAL(KIND=KIND(0.0D0)) :: a, b, c

DOUBLE PRECISION, DIMENSION(10) :: x, y, z

0.0D0, 7.0D0, 0.25D0, 1.23D0, 1.23D12,

0.1D0, 1.0D-1, 3.141592653589793D0.

INTEGER KIND

You can choose different sizes of integer

INTEGER, PARAMETER :: big = SELECTED_INT_KIND(12)

INTEGER(KIND=big) :: bignum

bignum can hold values of up to at least 1012

Example:

I = 45_big

Some compilers may allocate smaller integers

E.g. by using SELECTED_INT_KIND(4)

Using KIND

You should write and compile a module

```
MODULE accuracy
  INTEGER, PARAMETER :: rk =  SELECTED_REAL_KIND(12)
  INTEGER, PARAMETER :: ik =  SELECTED_REAL_KIND(12)
END MODULE accuracy
```

Immediately after every procedure statement

I.e. PROGRAM, SUBROUTINE or FUNCTION

```
USE accuracy
IMPLICIT NONE
real(rk) :: f,g
integer(ik) :: j,i
…
F = 45.0_dp ; g = cos(f)
j = 45_ik ; i = 56
```

Logical Type

These can take only two values: true or false

.TRUE. and .FALSE.

• Their type is LOGICAL

```
LOGICAL :: red, amber, green
IF (red) THEN
  PRINT *, 'Stop'
  red = .False. ; amber = .True. ; green = .False.
ELSIF (red .AND. amber) THEN
. . .
```

Exercise

Write a Fortran logical expression depending on five integers n1, n2, m1, m2 and k, which will be true if (and only if) the absolute magnitude of the difference between n1 and n2 exceeds that between m1 and m2 by at least the magnitude of k.

Logical Expressions

- Consists of one or more logical operators and logical, numeric or relational operands
  - values are .TRUE. or .FALSE.
  - Operators:

Operator	Example	Meaning
.AND.	A .AND. B	logical AND
.OR.	A .OR. B	logical OR
.NEQV.	A .NEQV. B	logical inequivalence
.XOR.	A .XOR. B	exclusive OR (same as .NEQV.)
.EQV.	A .EQV. B	logical equivalence
.NOT.	.NOT. A	logical negation

- Need to consider overall operator precedence (next slide)

Exercise: If gum1 and gum2 are logical variables both with the value **.TRUE.**, what are the values of

(i) gum = gum1.NEQV.gum2.EQV. .NOT.gum1;

(ii) gum = (gum1.NEQV.gum2) .EQV. .NOT.gum1;

(iii) gum = gum1.OR. .NOT.gum2 .NEQV.gum1.AND.gum2.

(iv) What is the value of (iii) above if gum1 is .TRUE. and.gum2 is .FALSE.?

Character Type

Used when strings of characters are required

Names, descriptions, headings, etc.

• Fortran's basic type is a fixed-length string

Unlike almost all more recent languages

• Character constants are quoted strings

```
PRINT *, 'This is a title'
PRINT *, "And so is this"
```

The characters between quotes are the value. . .

Character Constants

```
"This has UPPER, lower and MiXed cases"
’This has a double quote (") character’
"Apostrophe (’) is used for single quote"
"Quotes ("") are escaped by doubling"
’Apostrophes (’’) are escaped by doubling’
’ASCII ‘, |, ~, ^, @, # and \ are allowed here’

"Implementations may do non-standard things"
’Backslash (\) MAY need to be doubled’
"Avoid newlines, tabs etc. for your own sanity"
```

Character Variables

```
CHARACTER :: answer, marital_status

CHARACTER(LEN=10) :: name, dept, faculty

CHARACTER(LEN=32) :: address
```

answer and marital_status are each of length 1

They hold precisely one character each answer might be blank, or hold ’Y’ or ’N’

name, dept and faculty are of length 10

and address is of length 32

Exercise

Write type declaration statements to declare

(i) Three strings, each of length 24, called **s1, s2** and **s3;**

(ii) A character-string constant named me whose value is your surname;

(iii) A named character constant of length 1 called bs whose value is the backslash character (\).

Exercise: Reconstruct the string E

A="r astePear" (7:8)

B= " r astePear " (1:4)

C=" r astePear " (6:6)

D=" r astePear " (10:9)

E = A // B // C//D

Another Form

```
CHARACTER :: answer*1, marital_status*1, name*10
CHARACTER :: dept*10, faculty*10, address*32
```

While this form is historical, it is more compact

- Don’t mix the forms – this is an abomination

CHARACTER(LEN=10) :: dept, faculty, addr*32

- For obscure reasons, using LEN= is cleaner

Character Assignment

```
CHARACTER(LEN=6) :: forename, surname
  forename = 'Nick'
  surname = 'Maclaren'
```

forename is padded with spaces ('Nick ')

surname is truncated to fit ('Maclar')

- Unfortunately, you won't get told

But at least it won't overwrite something else

Character Concatenation

Values may be joined using the / / operator

```
CHARACTER(LEN=6) :: identity, A, B, Z
 identity = 'TH' // 'OMAS'
 A = 'TH' ; B = 'OMAS'
 Z = A // B
 PRINT *, Z
```

Sets identity to 'THOMAS'

But Z looks as if it is still 'TH' – why?

/ / does not remove trailing spaces

It uses the whole length of its inputs

Substrings

If Name has length 9 and holds ’Marmaduke’

Name(1:1) would refer to ’M’

Name(2:4) would refer to ’arm’

Name(6:) would refer to ’duke’ – note the form!

We could therefore write statements such as

```
CHARACTER :: name*20, surname*18, title*4
name = ’Dame Edna Everage’
title = name(1:4)
surname = name(11:)
```

message.f90

Example

```
PROGRAM message
  IMPLICIT NONE
  CHARACTER :: mess*72, date*14, name*40
    mess = ’Program run on’
    mess(30:) = ’by’
    READ *, date, name
    mess(16:29) = date
    mess(33:) = name
   PRINT *, mess
END PROGRAM message
```

```
$ 06.11.2012 Sergey
 Program run on 06.11.2012    by Sergey
$
```

Warning

CHARACTER substrings look like array sections

But there is no equivalent of array indexing

```
CHARACTER :: name*20, temp*1

temp = name(10)
```

- name(10) is an implicit function call

Use name(10:10) to get the tenth character

CHARACTER variables come in various lengths

name is not made up of 20 variables of length 1

Character Intrinsics

```
LEN(c)              ! The STORAGE length of c
TRIM(c)             ! c without trailing blanks
ADJUSTL(C)          ! With leading blanks removed
INDEX(str,sub)      ! Position of sub in str
SCAN(str,set)       ! Position of any character in set
REPEAT(str,num)     ! num copies of str, joined
```

Examples

```
name = ’ Bloggs ’
newname = TRIM(ADJUSTL(name))
```

newname would contain ’Bloggs’

```
CHARACTER(LEN=6) :: A, B, Z
A = ’TH’ ; B = ’OMAS’
Z = TRIM(A) // B
```

Now Z gets set to ’THOMAS’ correctly!

BUILT IN FUNCTIONS

Built in operators

- + Addition
  2+17 X+Y A+B+C+D
- - Subtraction or negation
  Income-Expenses A-B-C -X
- * Multiplication
  2*X Length*Area / Division
  I/J
  (X+Y)/Z
- ** Exponentiation
  2**5 P**Q (X+1)**2

Built in functions

Abs(X)	Absolute value of X.
Cos(X)	The cosine of X.
Exp(X)	The exponential function of X.
Int(X)	Makes an Integer copy of the Real number X. Int(1.9) is 1, for example.
Max(X1,X2)	The maximum of X1 and X2.
Min(X1,X2)	The minimum of X1 and X2.
Mod(X1,X2)	The remainder, when X1 is divided by X2. Mod(7,2) is 1, because 7 is odd. Mod(14.3, 3.0) is 2.3 because 14.3 = 3*4 + 2.3.
Nint(X)	Returns the nearest integer value. Nint(1.9) is 2, for example.
Real(I)	Make a Real copy of the Integer I.
Sign(X1,X2)	A value having the magnitude of X1, and the sign of X2. Sign(20.0, -7.0) is -20.0, for example.
Sin(X)	The sine of X. Sqrt(X) The square root of X.
Tan(X)	The tangent of X, sine(X)/cosine(X).
...	...

Math Functions

- sine and cosine (radians)
  - SIN(*real or double*) the generic version
  - SIN(*real*)
  - DSIN(*double*)
  - CSIN(*complex*)
- exponential
  - EXP(*real or double*) the generic version
  - EXP(*real*)
  - DEXP(*double*)
  - CEXP(*complex*)
- natural logarithm
  - LOG(*real or double*) the generic version
  - ALOG(*real*)
  - DLOG(*double*)
  - CLOG(*complex*)

Math Functions

- tangent (radians)
  - TAN(*real or double*) the generic version
  - TAN(*real*)
  - DSIN(*double*)
- square root
  - SQRT(*real or double*) the generic version
  - SQRT(*real*)
  - DSQRT(*double*)
  - CSQRT(*complex*)
- hyperbolic sine
  - SINH(*real or double*) the generic version
  - SINH(*real*)
  - DSINH(*double*)

Math Functions

- there are also similar functions for
  - arcsine, arccosine, arctangent (ASIN, ACOS, ATAN)
  - hyperbolic sine, cosine, tangent (SINH, COSH, TANH)
  - complex conjugate (CONJ)
  - base10 logarithms (LOG10)

Six operators which can be used to test numeric data

- `X .Eq. Y True if X equals Y.`
- `X .Ne. Y True if X is not equal to Y.`
- `X .Lt. Y True if X is less than Y.`
- `X .Le. Y True if X is less than or equal to Y.`
- `X .Gt. Y True if X is greater than Y.`
- `X .Ge. Y True if X is greater than or equal to Y.`

```
Z = X .Ge. Y  ! Logical result -> logical variable Z
```

Character Expressions

- Only built-in operator is Concatenation
  - defined by `//` - `'ILL'//'-'//'ADVISED'`

CODE

```
CHARACTER FAMILY*16
FAMILY = 'GEORGE P. BURDELL'
PRINT*,FAMILY(:6)
PRINT*,FAMILY(8:9)
PRINT*,FAMILY(11:)
PRINT*,FAMILY(:6)//FAMILY(10:)
```

OUTPUT

```
GEORGE
P.
BURDELL
GEORGE BURDELL
```

FUNCTIONS AND SUBROUTINES

Subroutines and functions

```
FUNCTION Variance (Array)
  IMPLICIT NONE
  REAL :: Variance, X
  REAL, INTENT(IN), DIMENSION(:) :: Array
  X = SUM(Array)/SIZE(Array)
  Variance = SUM((Array--X)**2)/SIZE(Array)
  END FUNCTION Variance
  REAL, DIMENSION(1000) :: data
  . . .
  Z = Variance(data)
  !We shall see how to declare it later
```

Example – Sorting

Replace the actual sorting code by a call

```
PROGRAM sort11
  IMPLICIT NONE
  INTEGER, DIMENSION(1:10) :: nums
   . . .
  ! ------ Sort the numbers into ascending order of magnitude
  CALL SORTIT (nums)
  ! ------ Write out the sorted list
  . . .
END PROGRAM sort11
```

Example – Sorting

```
SUBROUTINE SORTIT (array)
 IMPLICIT NONE
 INTEGER :: temp, array(:), J, K
  L1: DO J = 1, UBOUND(array,1)-1
   L2: DO K = J+1, UBOUND(array,1)
    IF(array(J) > array(K)) THEN
     temp = array(K)
     array(K) = array(J)
     array(J) = temp
    END IF
   END DO L2
  END DO L1
END SUBROUTINE SORTIT
```

SUBROUTINE Statement

Declares the procedure and its arguments

These are called dummy arguments in Fortran

The subroutine's interface is defined by:

- The SUBROUTINE statement itself
- The declarations of its dummy arguments
- And anything that those use (see later)

```
SUBROUTINE SORTIT (array)
INTEGER :: [ temp, ] array(:) [ , J, K ]
```

Subroutines With No Arguments

You aren't required to have any arguments

You can omit the parentheses if you prefer

Preferably either do or don't, but you can mix uses

```
SUBROUTINE Joe ( )
SUBROUTINE Joe
CALL Joe ( )
CALL Joe
```

Statement Order

A SUBROUTINE statement starts a subroutine

Any USE statements must come next

Then IMPLICIT NONE

Then the rest of the declarations

Then the executable statements

It ends at an END SUBROUTINE statement

PROGRAM and FUNCTION are similar

There are other rules, but you may ignore them

Functions

Often the required result is a single value

It is easier to write a FUNCTION subprogram

E.g. to find the largest of three values:

- Find the largest of the first and second
- Find the largest of that and the third

Yes, I know that the MAX function does this!

The function name defines a local variable

- Its value on return is the result returned

The RETURN statement does not take a value

Functions: example

```
FUNCTION largest_of (first, second, third)
 IMPLICIT NONE
 INTEGER :: largest_of
 INTEGER :: first, second, third
  !
  IF (first > second) THEN
   largest_of = first
  ELSE
   largest_of = second
  END IF
  IF (third > largest_of) largest_of = third
  !
END FUNCTION largest_of
```

Functions: example

```
program largest
 INTEGER :: trial1, trial2 ,trial3, total, count
 !
 total = 0 ; count = 0
 DO
   PRINT *, 'Type three trial values:'
   READ *, trial1, trial2, trial3
   IF (MIN(trial1, trial2, trial3) < 0) EXIT
      count = count + 1
      total = total + Largest_of(trial1, trial2, trial3)
   END DO
   PRINT *, 'Number of trial sets = ', count, &
  ' Total of best of 3 = ',total
end program largest
```

Functions With No Arguments

You aren’t required to have any arguments

You must not omit the parentheses

```
FUNCTION Fred ( )
INTEGER :: Fred
X = 1.23 * Fred ( )
CALL Alf ( Fred ( ) )
```

In the following, Fred is a procedure argument

```
CALL Alf ( Fred )
```

Recursive functions and routines

```
program factorial_example

 integer :: factorial,n

 print *," Inter an integer number"
 !
 read *,n
 !
 write(*,"('Factorial of ',i3,' = ',i7)") n,factorial(n)

end program factorial_example


recursive function factorial( n ) result( f )
  integer f
  integer, intent( in ) :: n

  if ( n <= 0 ) then
    f = 1
  else
    f = n * factorial( n-1 )
  end if
end function factorial
```

Example: Sierpinski tree

```
program tree

  implicit none
  !
  ! Variables
  !
  real(8), parameter :: pi    = 4.0d0 * atan2(1.0d0,1.0d0)
  real(8) :: angleFactor = pi/4.d0
  real(8) :: sizeFactor = 0.7 ! 0.592d0
  real(8) :: trunkHeight = 0.4d0
  integer(4) :: depth = 12

  ! Body of tree
  !
  call growTree(0.5d0,0.0d0,trunkHeight,pi/2.d0,depth,angleFactor,sizeFactor)
  !
end program tree

recursive subroutine growTree(x1,y1,rootLength,rootAngle,depth,angleFactor,sizeFactor)
  real(8),intent(in) :: x1,y1,rootLength,rootAngle,angleFactor,sizeFactor
  integer(4) :: depth
  real(8)    :: x2,y2
    !
    x2  = x1 + cos(rootAngle)*rootLength
    y2  = y1 + sin(rootAngle)*rootLength
!
    write(*,"('A ',3f12.8,1x,i4)")  x2,y2,0.0d0,depth
    !
    if (depth>0) then
      !
      call growTree(x2,y2,rootLength*sizeFactor,rootAngle+angleFactor,depth-1,angleFactor,sizeFactor)
      call growTree(x2,y2,rootLength*sizeFactor,rootAngle-angleFactor,depth-1,angleFactor,sizeFactor)
      !
    endif
    !
end  subroutine growTree
```

Using Modules

This is how to compile procedures separately

First create a file (e.g. mymod.f90) like:

```
MODULE mymod
 CONTAINS
FUNCTION Variance (Array)
  REAL :: Variance, X
  REAL, INTENT(IN), DIMENSION(:) :: Array
    X = SUM(Array)/SIZE(Array)
    Variance = SUM((Array-X)**2)/SIZE(Array)
  END FUNCTION Variance
END MODULE mymod
```

Using Modules

The module name need not be the file name

Doing that is strongly recommended, though

- You can include any number of procedures

You now compile it, but don't link it

gfortan -c mymod.f90

It will create files like mymod.mod and mymod.o

They contain the interface and the code

Using Modules

You use it in the following way

- You can use any number of modules

```
PROGRAM main
  USE mymod
    REAL, DIMENSION(10) :: array
    PRINT *, 'Type 10 values'
    READ *, array
    PRINT *, 'Variance = ', Variance(array)
END PROGRAM main
```

```
$gfortran main.f90 mymod.o  -o main.x
```