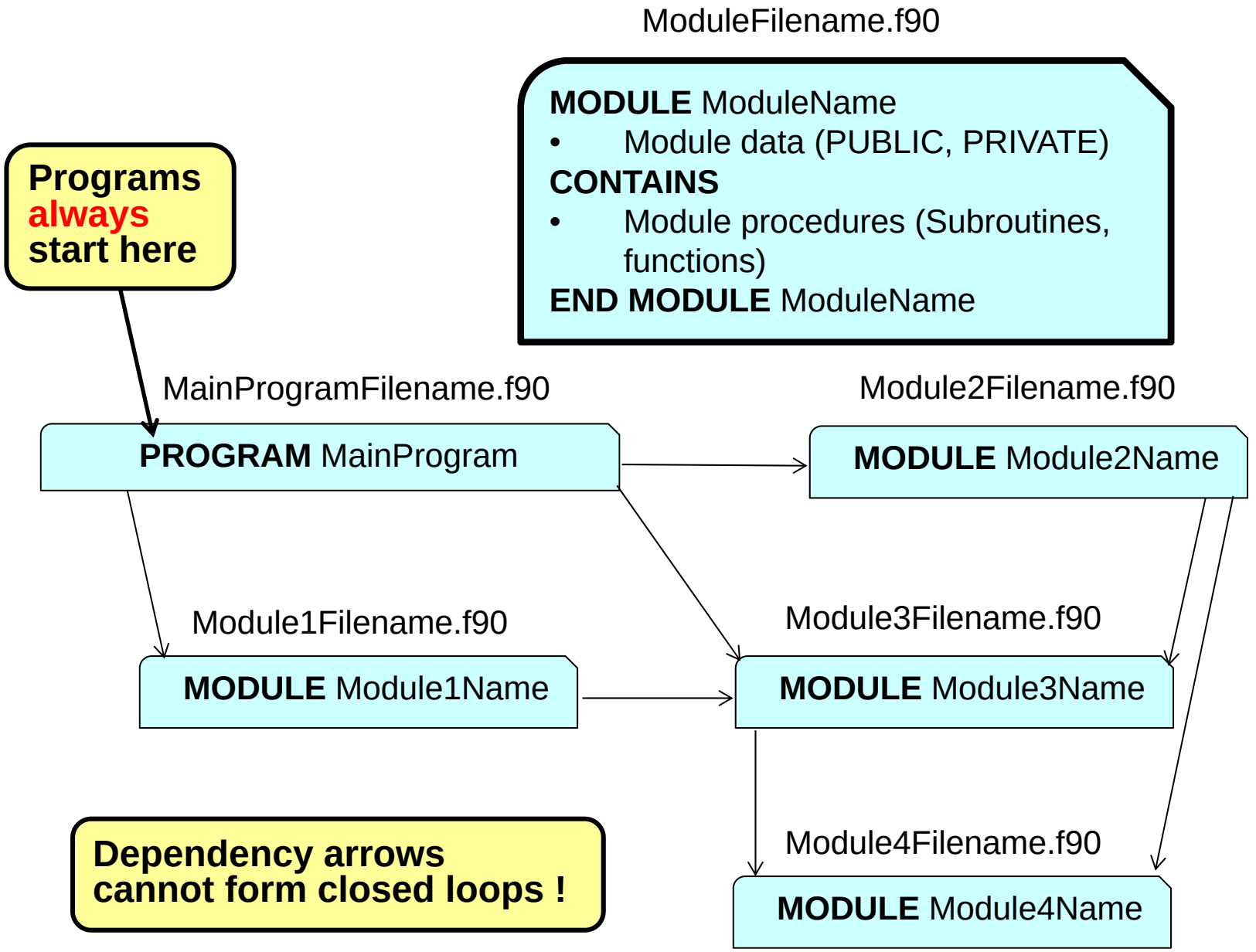
ECE 6350

Brief Introduction to Fortran 90/95

D. R. Wilton

University of Houston

High Level Overview of an O/O F90 Program



General Structure of an F90 Program

FileName.f90

PROGRAM statement with [optional] name

Declare modules (objects) and their data used

Ignore F77 default type conventions

Declare variable types, names, dimensions

-Start executable statements-

WRITE(*,*) – write to screen, unformatted
READ(*,*) – read from screen, unformatted

Stop program execution

End-of-program statement

```
PROGRAM [MainProgramName]
  !
  USE Object1[, ONLY:  Object1Variable &
                      ,Object1Procedure]
  IMPLICIT NONE
  !
  INTEGER                    :: N
  REAL, ALLOCATABLE  :: a(:)
  !
   N = 10
   ALLOCATE (a(N))
   CALL Test(a,N)
   WRITE(*,*) a
   WRITE(*,*) “Hello world”
   DEALLOCATE (a)
   STOP
   !
END PROGRAM [MainProgramName]
```

Variables

- **Names are up to 32 characters long**
- **Case is ignored**
- **Declare variables as one of these *types*:**
  - **REAL (1.0, -3.5E-9, etc.)**
  - **INTEGER (0, 3, -7, 2357, etc.)**
  - **CHARACTER (‘A’ – ‘Z’)**
  - **LOGICAL (.true., .false.)**
  - **COMPLEX ((1.0, -3.5E-2), (0.,1.0), etc.)**
  - **TYPE MyType ! *User-defined type with components that are one of the above or another user-defined type***
- **Optional variable *attributes* include**
  PARAMETER, ALLOCATABLE, POINTER, TARGET, SAVE, *or* INTENT(IN, OUT, INOUT)
- **Always use IMPLICIT NONE statement to ensure *strong* typing (i.e., to disallow *default* types)**

Variable Declaration

```
! Begin declaration section ("!" starts comment to end of line)
IMPLICIT NONE              ! Ignore F77 default type conventions
REAL [,optional*]          :: pi, a(0:9,0:9), b(:,:), c(:), d(10) ! Real arrays
INTEGER                    :: i, j, n
CHARACTER (Len = 6)        :: Username    !  Up to 6 characters
LOGICAL                    ::  SwitchOn    ! '.true.' or '.false.'
COMPLEX, PARAMETER :: complex_j = (0.,1.)
TYPE LayerType              ! LayerType is a user defined structure
      REAL                 :: Thickness
      COMPLEX              :: RelativePermittivity, RelativePermeability
END TYPE LayerType
! Define variable of type LayerType
TYPE(LayerType) :: Layer(10) ! Layer is a 10 element array
! End declaration section
 ...
Layer(1)%Thickness  = 2.0        ! Set components of Layer(1)
Layer(1)%RelativePermittivity = ( 2.0, - 0.01 )
Layer(1)% RelativePermeability = ( 1.0,  0.0 )
Layer(3) = Layer(1)      ! Equate all components of Layer(3) and Layer(1)
```

or **Layer(1)**
= LayerType(2.0, (2.0,-0.01), (1.0, 0.0))

***optional** = PARAMETER, ALLOCATABLE, POINTER, TARGET, SAVE, **or** INTENT(IN,OUT,INOUT)*

A Simple Loop

```
[MyLoop: &]          ! Name "MyLoop" helps find its end
DO i = 1, 5 [, 2]    ! i starts at 1, goes to 5 [with stride 2]
  WRITE(*,*) i,  2*i, i/2, i+2,  i**2  !writes i, 2i, ⌊i/2⌋, i+2, i^2
END DO [MyLoop]      ! Finally, the end of "MyLoop"
```

& = indicates continuation onto the next line

! = "comment" from here to end of line

(*,*) = (output to screen, unformatted output)

[ ] = optional parameter

Branching with IF Statements

Executable statement

single alternative IF

```
IF(SwitchOn == .TRUE.) a=b
```

double alternative IF

```
!
IF(SwitchOn == .TRUE.) THEN
   First set of executable statements
ELSE
   Second set of executable statements
END IF
```

multi - alternative IF

```
!
IF(i == 0) THEN
   First set of executable statements
ELSEIF(i  > 0) THEN
   Second set of executable statements
ELSE     ! At this point, i<0
   Third set of executable statements
END IF
```

Logical Compare:

/=	"not equal"
==	"equal"
<	"less than"
<=	"less than or equal"
>	"greater than"
>=	"greater than or equal"

Branching with Case Statements

```
SELECT CASE (MyVariable)
!
 CASE(Value1)                    ! MyVariable=Value1
     First set of executable statements
 CASE(Value2)                    ! MyVariable=Value2
     Second set of executable statements
 CASE(Value3)                    ! MyVariable=Value3
     Third set of executable statements
 CASE DEFAULT
     Default executable statements
!
END SELECT
```

A Simple O/O F95 Code with One "Object"

In F90, *modules* are intended to encapsulate objects

File Main.f90

```
PROGRAM Main
  USE Object1, ONLY: a,Test
  IMPLICIT NONE
  INTEGER :: N
  N=10
  CALL Test(a,N)
  WRITE(*,*) a
  WRITE(*,*) "Hello world"
  DEALLOCATE (a)
  STOP
END PROGRAM Main
```

Object1Data

Object1 Procedures

File Object1.f90

```
MODULE Object1
  IMPLICIT NONE
  REAL, ALLOCATABLE :: a(:,:)
!
  CONTAINS
!=======================
  SUBROUTINE Test(d,N)
   IMPLICIT NONE
   INTEGER :: i,j,N
   REAL, ALLOCATABLE ::d(:,:)
   ALLOCATE (d(N,N))
     DO i = 1, SIZE(d,DIM=1)
        DO j = 1, SIZE(d,DIM=2)
          d(i,j) = i-j   ! Fill array
        END DO
     END DO
  END SUBROUTINE Test
!=======================
END MODULE Object1
```

"…ONLY: a,Test" is optional, but ensures we can trace definitions of *a* and *Test*!

Variable "*d*" in *Test* is a "dummy" variable

Simple Changes So That *Test* Uses Its Own Object Data

File Main.f90

```
PROGRAM Main
  USE Object1, ONLY: a,Test
  IMPLICIT NONE
  INTEGER :: N
  N=10
  CALL Test(N)
  WRITE(*,*) a
  WRITE(*,*) “Hello world”
  DEALLOCATE (a)
  STOP
END PROGRAM Main
```

Program Main does not now have to pass data a(:,:)to Test

Object Data

Object Procedures

File Object1.f90

```
MODULE Object1
  IMPLICIT NONE
  REAL, ALLOCATABLE :: a(:,:)
!
  CONTAINS
!=======================
  SUBROUTINE Test(N)
   IMPLICIT NONE
   INTEGER :: i,j,N
     ALLOCATE (a(N,N))
     DO i = 1, SIZE(a,DIM=1)
        DO j = 1, SIZE(a,DIM=2)
          a(i,j) = i-j   ! Fill array
        END DO
     END DO
  END SUBROUTINE Test
!=======================
END MODULE Object1
```

Procedure Test can now access Object1’s data, a(:,:)

The End