Derived Data Types

**Intrinsic Data Types:**

- INTEGER, REAL, COMPLEX, LOGICAL,
- CHARACTER, POINTER

**Derived Data Types:**

- also called user defined data types
- must be derived from the intrinsic
data types and/or previously defined new data types.
- must be defined before objects of the derived type can be declared.

A derived type definition specifies the name of the new type and the names and types of its components.

**Syntax:**

```
TYPE :: type_name
    component_definition
  :
  :
END TYPE type_name
```

**Example 1:**

```
TYPE :: student
    CHARACTER :: first_name*12, middle_init*1, last_name*12
    REAL :: grade_point_avg
    CHARACTER (LEN=11) :: ssn
END TYPE student
```
Examples: Derived Data Types

Example 2:

```fortran
TYPE :: coordinates
  REAL :: x, y
END TYPE coordinates
```

Example 3:

```fortran
TYPE :: person
  CHARACTER (LEN=12) :: first_name, last_name
  CHARACTER (LEN=11) :: ssn
END TYPE person
```

Declaring Variables of Derived Type

Variables of derived types are declared with the TYPE statement.

Examples:

```fortran
TYPE (student) :: john
TYPE (coordinates) :: origin
```

It is allowed to use a previously defined derived type in the definition of another derived type:

```fortran
TYPE :: employee
  TYPE (person) :: employee
  CHARACTER (LEN=20) :: department
  REAL :: salary
END TYPE employee
```

```fortran
TYPE (employee) :: smith
```
Examples (cont’d)

Component References

Components are referenced using the percent sign ‘%’ operator. For example, last_name in the structure john is referenced as john%last_name. Similarly, ssn in type employee in structure smith is referenced as smith%employee%ssn.

john%last_name = “Foster”
smith%employee%ssn = “123-34-6745”
origin%x = origin%x + 25.5
origin%y = origin%y - 12.3
READ(*,*) origin%x, john%first_name
WRITE(*,*) smith%employee%ssn

Structure Constructors

Syntax:

type-name (list of expressions)

A structure constructor is a form of defining a constant value for a derived type.

Examples:

john = student(“John”, ”M”, ”Foster”, 3.9, ”234-34-2345”)
WRITE(*,*) john ! write sequence of all components
READ(*,*) john ! read values for all components
! Modified example 3.3, Text page 71 (Ellis, Phillips, & Lahey)

PROGRAM geometry

IMPLICIT NONE

! A program to use derived types for two-dimensional
! geometric calculations

! Type definitions

TYPE :: point
  REAL :: x, y       ! Cartesian coordinates of the point
END TYPE point

TYPE :: line
  REAL :: a, b, c   ! coefficients of defining equation
END TYPE line

! Variable declarations

TYPE(point) :: p1, p2
TYPE(line)   :: p1_to_p2

! Read data

WRITE(*,*) "Please type coordinates of first point"

READ (*,*) p1
WRITE (*,*) "Please type coordinates of second point"

READ (*,*) p2

! Calculate coefficients of equation representing the line
p1_to_p2%a = p2%y - p1%y
p1_to_p2%b = p1%x - p2%x
p1_to_p2%c = p1%y*p2%x - p2%y*p1%x

! Print result
WRITE (*,*) "The equation of the line joining these two points is"
WRITE (*,*) "ax + by + c = 0"
WRITE (*,*) "where a = ", p1_to_p2%a
WRITE (*,*) "  b = ", p1_to_p2%b
WRITE (*,*) "  c = ", p1_to_p2%c
STOP
END PROGRAM geometry
Examples (cont’d)

! Modified Example 3.4, Text page 73 (Ellis, Phillips, & Lahey)

PROGRAM complex_arithmetic
    IMPLICIT NONE

    ! A program to illustrate the use of a derived type to perform
    ! complex arithmetic

    ! Type definition
    TYPE :: complex_number
        REAL :: real_part, imaginary_part
    END TYPE complex_number

    ! Variable definitions
    TYPE(complex_number) :: c1, c2, sum, diff, prod

    ! Read data
    WRITE (*,*) "Please supply two complex numbers"
    WRITE (*,*) "Each complex number should be typed as two numbers, "
    WRITE (*,*) "representing the real and imaginary parts &
        &of the number"
READ (*,*) c1, c2

! Calculate sum, difference and product

sum%real_part = c1%real_part + c2%real_part
sum%imaginary_part = c1%imaginary_part + c2%imaginary_part
diff%real_part = c1%real_part - c2%real_part
diff%imaginary_part = c1%imaginary_part - c2%imaginary_part

prod%real_part = c1%real_part * c2%real_part - &
               c1%imaginary_part * c2%imaginary_part

prod%imaginary_part = c1%real_part * c2%imaginary_part + &
                       c1%imaginary_part * c2%real_part

! Print results
WRITE (*,*) "The sum of the two numbers is ", sum
WRITE (*,*) "The difference between the two numbers is ", diff
WRITE (*,*) "The product of the two numbers is ", prod
STOP
END PROGRAM complex_arithmetic