

1. Subprograms (functions & subroutines): Reusable code segments

Up until now, all code has been written inside the main *program*. Fortran also has ability to write independent pieces of code that can be called repetitively and/or by different programs and subprograms. Subprogram (AKA *procedure* is a generic term for these code segments, and F90 has two types of subprograms:

- *Functions*: return a value by their invocation, and can be used in expressions
- *Subroutines*: are called with the CALL command, and pass any needed output through arguments

Advantages of subprograms:

- Possibility of reusing same code section without duplicating code
- Improved design, development and debugging through increased *modularity*.
- Greater abstraction of tasks by using *dummy arguments*

2. Procedure-related statements and definitions

- **actual arguments**: values/variables that procedure is invoked with (i.e., the values to be loaded into the dummy argument names)
- **dummy arguments**: internal name that procedure calls its arguments
 - **INTENT(IN)**: specifies dummy argument brings in input from caller, but will not be changed or return a value to caller
 - **INTENT(OUT)**: initial value of dummy arg will not be read by function – dummy arg is for returning value(s) to caller only.
 - **INTENT(INOUT)**: specifies dummy argument will be used for both input into, and returning values from, the procedure
- **CALL**: invokes a subroutine
- **RETURN**: stops executing function and returns to invocation site
- **EXTERNAL**: indicates that the procedure is external to the present compilation scope

3. Pure Functions

Pure Functions take any number of input arguments, and return one output argument (like built-in functions such as SUM, SIZE, or MINVAL).

<p>General outline</p> <pre>PROGRAM progran ... variable = funcnam[(actual_arg_list)] ... END PROGRAM progran FUNCTION funcnam[(dummy_arg_list)] funcnam \& arg declarations variable declarations executable statements END FUNCTION funcnam</pre>	<p>Function to compute area of a rectangle</p> <pre>PROGRAM getAreas REAL :: L, W, area EXTERNAL :: area_rect REAL :: area_rect DO PRINT *, 'Enter length & width, separated b READ(*,*) L, W IF (L .LT. 0.0 .OR. W .LT. 0.0) EXIT area = area_rect(L,W) PRINT *, 'AREA is ', area END DO END PROGRAM getAreas FUNCTION area_rect(length, width) REAL :: area_rect REAL, INTENT(IN) :: length, width area_rect = length * width END FUNCTION area_rect</pre>
---	---

4. Impure Functions

Functions that return values other than through their invocation are not pure. Here's an example

<pre>PROGRAM getAreas IMPLICIT NONE REAL :: L, W, area EXTERNAL :: area_rect, GetGoodInput REAL :: area_rect LOGICAL :: GetGoodInput DO WHILE(GetGoodInput(L,W)) area = area_rect(L,W) PRINT *, 'AREA is ', area END DO END PROGRAM getAreas FUNCTION area_rect(length, width) ! Pure function, returning area IMPLICIT NONE REAL :: area_rect REAL, INTENT(IN) :: length, width area_rect = length * width END FUNCTION area_rect</pre>	<p>Impure function</p> <pre>FUNCTION GetGoodInput(length, width) ! ! This function prompts the user for ! length and width, and returns .TRUE. ! if the input is good (>= 0) and ! .FALSE. if the input is bad ! IMPLICIT NONE LOGICAL :: GetGoodInput REAL, INTENT(OUT) :: length, width GetGoodInput = .FALSE. PRINT *, 'Enter length: ' READ(*,*) length IF (length .LT. 0.0) RETURN PRINT *, 'Enter width : ' READ(*,*) width IF (width .LT. 0.0) RETURN GetGoodInput = .TRUE. END FUNCTION GetGoodInput</pre>
--	---

5. Subroutines

A *subroutine* is a subprogram that returns its output only by using arguments. It is invoked using the CALL statement.

General outline

```
PROGRAM program
  call subnam[(actual_arg_list)]
END PROGRAM program
SUBROUTINE subnam[(dummy_arg_list)]
  arg declarations
  variable declarations
  executable statements
END SUBROUTINE subnam
```

Main program

```
PROGRAM getAreas
  IMPLICIT NONE
  REAL :: L, W, area
  LOGICAL :: KeepOn
  EXTERNAL :: area_of_rect, GetInput
  DO
    CALL GetInput(KeepOn, L, W)
    IF (.NOT.KeepOn) EXIT
    CALL area_of_rect(L, W, area)
    PRINT *, 'AREA is ', area
  END DO
END PROGRAM getAreas
```

Subroutines to compute areas of a rectangle

```
SUBROUTINE GetInput(GoodInput, length, width)
  IMPLICIT NONE
  LOGICAL, INTENT(OUT) :: GoodInput
  REAL, INTENT(OUT) :: length, width

  GoodInput = .FALSE.
  PRINT *, 'Enter length: '
  READ(*,*) length
  IF (length .LT. 0.0) RETURN
  PRINT *, 'Enter width : '
  READ(*,*) width
  IF (width .LT. 0.0) RETURN
  GoodInput = .TRUE.
END SUBROUTINE GetInput

SUBROUTINE area_of_rect(length, width, area)
  IMPLICIT NONE
  REAL, INTENT(IN) :: length, width
  REAL, INTENT(OUT) :: area
  area = length * width
END SUBROUTINE area_of_rect
```

6. Dot Product Example

```
PROGRAM CheckDotProd
  IMPLICIT NONE
  REAL :: X(200), Y(600), Z(0:99)
  EXTERNAL :: dot_prod
  REAL :: dot_prod

  CALL RANDOM_NUMBER(X)
  CALL RANDOM_NUMBER(Y)
  CALL RANDOM_NUMBER(Z)

  PRINT *, 'X*Y = ', dot_prod(200,X,Y), &
    dot_product(X,Y(1:200))
  PRINT *, 'X*Z = ', dot_prod(100,X,Z), &
    dot_product(X(1:100),Z)
  PRINT *, 'Sec = ', &
    dot_prod(6, X(5:10), Z(3:8)), &
    dot_product(X(5:10), Z(3:8))
END PROGRAM CheckDotProd
```

```
FUNCTION dot_prod(N, X, Y)
  IMPLICIT NONE
  INTEGER :: N
  REAL :: dot_prod
  REAL, INTENT(IN) :: X(*), Y(*)
  INTEGER I

  dot_prod = 0.0
  IF (N .LT. 1) RETURN

  DO I = 1, N
    dot_prod = dot_prod + X(i) * Y(i)
  END DO
END FUNCTION dot_prod
```

7. Matrix add

For multidimensional arrays, need to tell it how large all but last dimensions are, so that compiler can do the correct indexing. Call the first dimension the *leading dimension*.

```
PROGRAM addtest
  INTEGER, PARAMETER :: lda=500, ldb=400, &
    M=350, N=300
  REAL :: A(lda,N), B(ldb,N), C(ldb,N)
  EXTERNAL :: matadd
  !
  ! Generate random A & B, and compute
  ! correct answer in C using subsections
  !
  CALL RANDOM_NUMBER(A)
  CALL RANDOM_NUMBER(B)
  C(1:M,1:N) = A(1:M,1:N) + B(1:M,1:N)
  CALL matadd(M, N, A, lda, B, ldb)
  !
  ! Error check
  !
  PRINT*, 'Residual = ', &
    maxval(abs(C(1:M,1:N) - B(1:M,1:N)))
END PROGRAM addtest

SUBROUTINE matadd(M, N, A, lda, B, ldb)
  !
  ! Returns B = A + B
  !
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: M, N, lda, ldb
  REAL, INTENT(IN) :: A(lda,*)
  REAL, INTENT(INOUT) :: B(ldb,*)
  INTEGER :: i, j

  DO j = 1, N
    DO i = 1, M
      B(i,j) = B(i,j) + A(i,j)
    END DO
  END DO
END SUBROUTINE matadd
```