Program Units

FORTRAN PROGRAM UNITS

1. The Main Program Unit
It begins with a PROGRAM statement and ends with an END PROGRAM statement.

```
PROGRAM  program_name
         :
         specification statements
         :
         executable statements
         :
END PROGRAM program_name
```

Note: Execution of a program always begins with the main program unit.

2. Function Subprograms

```
FUNCTION function_name (list_of_arguments)
         :
         specification statements
         :
         executable statements
         :
END FUNCTION function_name
```

Prorgram Units (cont'd)

3. Subroutine Subprograms

```
SUBROUTINE  subroutine_name(list_of_arguments)
        :
        specification statements
        :
        executable statements
        :
END SUBROUTINE subroutine_name
```

4. Module Program Units

```
MODULE module_name
        :
        specification statements
        :
        executable statements
        :
END MODULE module_name
```

5. Block Data Program Units

```
BLOCK DATA  block_name
        :
        specification statements
        :
END BLOCK DATA block_name
```

Examples

Example 1: Functions

A function is a subprogram that produces a single scalar or array result.

```
      PROGRAM main
      IMPLICIT NONE

      INTERFACE                          ! Required in ELF90
        FUNCTION square(x)
          IMPLICIT NONE
         REAL, INTENT(IN) :: x
               REAL  :: square
    END FUNCTION square
END INTERFACE

   REAL :: a, b = 2.3,  c = 3.5,  square
      a = 3.2 + b + square(c) + sin(2.5)
      WRITE(*,*) a
      STOP
      END PROGRAM main

      FUNCTION square(x)
        IMPLICIT NONE
        REAL, INTENT(IN) :: x
    REAL :: square
    square = x*x
    RETURN
 END FUNCTION square
```

Examples

Example 2: Subroutines

A subroutine is a subprogram that is invoked using a CALL statement.

```
PROGRAM main
IMPLICIT NONE
INTERFACE                          ! Required in ELF90
  SUBROUTINE multiply(x, y)
        IMPLICIT NONE
        REAL, INTENT(IN  OUT) :: x
        REAL, INTENT(IN) :: y
  END SUBROUTINE multiply
END INTERFACE

REAL :: a, b
a = 4.2
b = 10.5
CALL multiply(a, b)
WRITE (*,*) a
STOP
END PROGRAM main

SUBROUTINE multiply(x, y)
   IMPLICIT NONE
   REAL, INTENT(IN   OUT) :: x
   REAL, INTENT(IN) :: y
   x = x * y
END SUBROUTINE multiply
```

Arguments

Note: A Fortran procedure can be a function, a subroutine, or the main program unit. A reference to a function or a subroutine with arguments from a procedure causes transfer of control from the procedure to the function or to the subroutine. The process is called calling or invoking a function or a subroutine. At the end of processing the called function or the subroutine returns control to the calling procedure.

ARGUMENTS:
Arguments are used to pass information between a calling procedure and the called procedure. Arguments used in calling a function or a subroutine in a procedure are called actual arguments. Arguments specified in defining a function or a subroutine are called dummy arguments. Actual arguments can be variables, arrays, function references or expressions. Dummy arguments can be variables and arrays only.

There exists one-to-one correspondence between the acutal arguments and the dummy arguments. The number, order, and types of the actual arguments used in calling a procedure must exactly match with the number, order, and types of the corresponding dummy arguments in the called procedure.

Note: the compiler will check order and type correspondence but it is the programmer's responsibility to check order.

Argument Intent

Argument Intent

The INTENT attribute protects a program from undesirable side effects. The following are the three possible forms of the INTENT attribute for a dummy variable:

- **INTENT (IN)**
  Used to input data to the procedure.
  The procedure will not be able to alter its value.
  (See example 1)
- **INTENT (OUT)**
  Used to return the result.
- **INTENT (IN OUT)**
  Used for inputting and returning a result.
  (See example 2)

Local Variables

Variables declared in a program unit are only known to the unit itself. They have NO relationship to identically named variables declared elsewhere. Dummy arguments can have the same name as their corresponding passed arguments, (but are not required to be identical).

Functions

A FUNCTION statement in Fortran can be written in general form as:

FUNCTION func_name (d1, d2, …)

where d1, d2, … are dummy arguments.

- **A function must end with the END FUNCTION statement.**
- **There must be a declaration with function type.**
- **There must be at least one statement that assigns a value to the function name. A function returns only one value.**
- **ELF90 requires at least one RETURN statement in the function.**

Function Invocation

Invocation

To invoke a function, simply use its name (including the arguments) just as you would use as a regular variable name.

Examples:

```
q = func_name(a1, a2, …., an)
          :
WRITE(…) func_name(a1, a2, …., an)
          :
IF (func_name(a1, a2, …., an) .EQ. val ) THEN ….
```

Example 3

The following function returns the maximum value of two variables passed to the function.

```
FUNCTION  my_max (first, second)
  IMPLICIT NONE
 ! Declare types of the dummy arguments
 INTEGER, INTENT(IN) :: first, second

 INTEGER :: my_max            ! Function type
! Processing
IF (first > second) THEN
   my_max = first
ELSE
   my_max = second
END IF
RETURN
END FUNCTION my_max
```

Example 3 (Cont'd)

Elf90 requires an explicit interface for a function call from a procedure. The following code segment shows how the function can be invoked from the main program unit.

```
PROGRAM main
IMPLICIT NONE
INTERFACE
   FUNCTION my_max(x, y)
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: x, y
      INTEGER :: my_max
   END FUNCTION my_max
END INTERFACE

INTEGER :: a, b, c
WRITE(*,*) "Enter two integers:"
READ(*,*) a, b
WRITE(*,*) "The larger integer is: " , my_max(a,b)
    :
c = 67 + my_max(a+5, 2*b)
    :
c =my_max( my_max(a+4, b), 80)
    :
STOP
END PROGRAM  main
```

Subroutines

Subroutines

The general form of the SUBROUTINE statement is as follows:

SUBROUTINE sub_name (d1, d2, …)

where d1, d2 … are dummy arguments.

- **A subroutine must end with the END SUBROUTINE statement.**
- **A subroutine is invoked by the CALL statement:**

  CALL sub_name (arg1, arg2, …)
- **A subroutine need not return anything to the calling procedure.**
- **Results are returned thru the arguments (NOT in the name as in the case of a function). A subroutine does its work by changing the values of its arguments. This causes the same changes in the corresponding arguments in the calling program.**
- **Actual arguments can be almost anything: array names, simple variables, array elements, constants, expressions, function references, etc.**
- **Dummy arguments can only be simple variables or array names. The following subroutine heading would be a <u>syntax error.</u>**

SUBROUTINE sub2(x*y, 15, a(2, 3))

Examples (subroutine calls):

```
CALL sub1 (x, av, 15)
CALL sub2 (x * y, a, b, weeks)
q = fun1 (ara, len)
z = fun2 (ara, ara(3), last - 1)
CALL sub3 (fun1 (ara, len), b * c)
```

Note: Subroutines are CALLed. Functions occur in expressions.

Subroutine, Example

Example 4:
Write a subroutine to find the max, min, average, and sum of all the elements of any sizes real array.

Code:

```
! Main Program
PROGRAM STATS
IMPLICIT NONE
INTERFACE
 SUBROUTINE docalc(arr, size, max, min, avg, sum)
  IMPLICIT NONE
  REAL,DIMENSION(:),INTENT(IN) :: arr
  INTEGER, INTENT(IN) :: size
  REAL,INTENT(OUT) :: max, min, avg, sum
 END SUBROUTINE docalc
END INTERFACE
INTEGER :: i
REAL :: grades(50), iqs(50), gmax, gmin, gavg, gsum, iqmax, &
      iqmin, iqavg, iqsum

OPEN(9,”arr.dat”)
OPEN(10,”arr.out”)

DO i = 1, 50
 READ(9,*) grades(i), iqs(i)
END DO

CALL docalc(grades,50,gmax,gmin,gavg,gsum)
WRITE(10,*)gmax,gmin,gavg,gsum

CALL docalc(iqs,50,iqmax,iqmin,iqavg,iqsum)
WRITE(10,*)iqmax,iqmin,iqavg,iqsum

STOP
END PROGRAM stats
```

Subroutine, Example (Cont'd)

```
! The subroutine that calculates the max, min, avg, and sum

SUBROUTINE docalc(arr, size, max, min, avg, sum)
  IMPLICIT NONE
  REAL,DIMENSION(:),INTENT(IN) :: arr
  INTEGER, INTENT(IN) :: size
  REAL,INTENT(OUT) :: max, min, avg, sum
  INTEGER :: i
  max = arr(1)
  min = arr(1)
  sum = arr(1)
  DO i = 2, size
    sum = sum + arr(i)
    IF(arr(i) > max) max = arr(i)
    IF(arr(i) < min) min = arr(i)
  END DO
  avg = sum / size
  RETURN

END SUBROUTINE docalc
```

Internal Procedures

Internal Procedures

A procedure or the main program unit (called the host procedure) can contain other procedures within itself. All internal procedures are specified within the host procedure following a CONTAINS statement which must appear after all the executable code of the containing procedure. In ELF90, no INTERFACE statement is required to invoke an internal procedure from the host procedure.

Example:

```
SUBROUTINE  host_procedure_name(…….)
                  :
    CALL   internal_procedure_name (…….)
                  :
                  :
                  :
CONTAINS

    SUBROUTINE internal_procedure_name (……)
                  :
                  :
                  :
    END SUBROUTINE internal_procedure_name(…...)

END  SUBROUTINE host_procedure_name
```

Note: An internal procedure may be not accessible from any other procedure except the host procedure. An internal procedure has access to entities of its host.

Problem:

Read in a list of 100 numbers, print them out in ascending order.

The problem involves sorting of the numbers. Numerous sorting algorithms are known. Among them, the selection sort algorithm is easy to understand (but it is not the most efficient one).

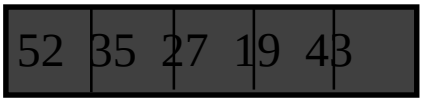
Selection Sort Trace

General Actions:

1. Find the smallest element in the array, beginning with the k^{th} element.
2. Exchange it with the k^{th} element

Trace Example:

Given the following array, go through the sort steps.



find smallest, beginning with 1st element
52 35 27 19 43 19 is the smallest
exchange it with 1st element
19 35 27 52 43
find smallest, beginning with 2nd element
19 35 27 52 43 27 is the smallest
exchange it with the 2nd element
19 27 35 52 43
find smallest, beginning with 3rd element
19 27 35 52 43 35 is the smallest
exchange it with the 3rd element
19 27 35 52 43
find smallest, beginning with 4th element
19 27 35 52 43 43 is the smallest
exchange it with the 4th element
19 27 35 43 52

Done! The array is sorted.

Note: the array contained 5 elements, but only the 4 smallest were found & moved. In general, once all of the array elements, but one, are swapped to their correct sorted positions, the remaining element must be in it's correct position.

Selection Sort Code

Code:

```
PROGRAM sortsample
  IMPLICIT NONE
  INTEGER, PARAMETER :: elems = 10
  INTEGER :: ints(elems), i
  DO i = 1, elems
    READ(*,*) ints(i)
  END DO
  CALL sort( )
  DO  i = 1, elems
    WRITE(*,*) ints(i)
  END DO
  STOP
CONTAINS
  SUBROUTINE sort( )     ! Sort routine
    INTEGER:: j, temp, lowsub
    DO i = 1, elems-1
      lowsub = i
      ! Find index of smallest in remaining elements
      DO j = i + 1, elems
        IF (ints(j) < ints(lowsub)) lowsub = j
      END DO
      ! Swap the two elements
      temp= ints(i)
      ints(i)= ints(lowsub)
      ints(lowsub) = temp
    END DO
    RETURN
    END SUBROUTINE sort
END PROGRAM sortsample
```

Why does it take three statements to do a swap?

Module Program Units

Module program units are used to package anything that is required by more than one program units or components. Typical uses of modules are:

1) Declaration and initialization of data.

2) Specification of explicit interfaces for procedures.

3) Definition of derived data types to be used in many procedures.

Refer to text section 4.7 for examples.