Procedures and Structured Programming

FORTRAN has a special mechanism designed to make subtasks easy to develop and debug independently before building the final program. It is possible to code each subtask as a separate program unit called an external procedure, and each external procedure can be compiled, tested, and debugged independently of all of the other subtasks (procedures) in the program.

FORTRAN has two kinds of external procedures: subroutines and function subprograms (or just functions).

Subroutines

A subroutine is a FORTRAN procedure that is invoked by naming it in a CALL statement and that receives its input values and returns its results through an argument list. General form is

```
SUBROUTINE subroutine_name (argument_list)
 ….
  (Declaration section)
….
 (Execution section)
….
RETURN
END SUBROUTINE [name]
```

The argument list contains a list of the variables and/or arrays that are being passed from the calling program to the subroutine. These variables are called as **dummy arguments**, since the subroutine does not actually allocate any memory for them. They are just placeholders for actual arguments that will be passed from the calling program unit when the subroutine is invoked.

Any executable program unit may call a subroutine, including another subroutine. To call a subroutine, the calling program uses a CALL statement. General form of CALL statement is

```
CALL subroutine_name (argument_list)
```

where the order and type of the **actual arguments** in the argument list must match the order and type of the dummy arguments declared in the subroutine.

The Intent Attribute

Dummy subroutine arguments can have an INTENT attribute associated with them. The INTENT attribute is associated with the type declaration statement that declares each dummy argument. The purpose of the INTENT attribute is to tell the compiler how the programmer intends to use each dummy argument. The attribute can take one of three forms:

INTENT (IN)	Dummy argument is used only to pass input data to the Subroutine.
INTENT (OUT)	Dummy argument is used only to return results to the Calling program.
INTENT (IN OUT)	Dummy argument is used both to pass input data to the subroutine and to return results to the calling program.

Example:

```
SUBROUTINE sub1(input, output)
IMPLICIT  NONE
REAL, INTENT(IN) :: input
REAL, INTENT(OUT) :: output
output = 2. * input
input = -1.                    ! This line is a error !
END SUBROUTINE
```

Always declare the intent of every dummy argument in every procedure.

FORTRAN provides a way to guarantee that local variables and arrays are saved unchanged between calls to procedure. This is the SAVE attribute. For example, a local variable sums could be declared with the SAVE attribute as

```
REAL, SAVE :: sums
```

The format of SAVE statement is

```
SAVE :: var1, var2, …
```

Or simply

```
SAVE
```

If a procedure requires that the value of a local variable not change between successive invocations of the procedure, include the SAVE attribute in the variable's type declaration statement, include the variable in a SAVE statement, or initialize the variable in its type declaration statement. If you do so, the subroutine will work correctly with some processors but will fail with others.

Example 1 of a module:

```
MODULE  test
IMPLICIT NONE
SAVE
INTEGER, PARAMETER :: num_vals = 5
REAL, DIMENSION(num_vals) :: values
END MODULE test

PROGRAM test_module
USE test
IMPLICIT NONE
REAL, PARAMETER :: pi = 3.141592     ! Pi
Values = pi * ( / 1., 2., 3., 4., 5. /)
 CALL sub1                                ! Call subroutine
END PROGRAM

SUBROUTINE sub1
USE test
IMPLICIT NONE
WRITE (*,*) values
END SUBROUTINE sub1
```

The contents of module test are being shared between the main program and subroutine sub1.

You may use a module to pass large amounts of data between procedures within a program. If you do so, always include the SAVE statement within the module to ensure that the contents of the module remain unchanged between uses. To access the data in a particular program unit, include a USE statement as the first noncomment statement after the PROGRAM, SUBROUTINE, or FUNCTION statement within the program unit.

Example 2,

```
MODULE my_subs
IMPLICIT NONE
(Declare shared data here)
CONTAINS
SUBROUTINE sub1( a, b, c, x, error)
IMPLICIT NONE
REAL, DIMENSION(3), INTENT(IN) :: a
REAL, INTENT (IN) :: b,c
REAL, INTENT (OUT) :: x
LOGICAL, INTENT (OUT) :: error
….
END SUBROUTINE sub1
END MODULE my_subs

PROGRAM main_prog
USE my_subs
IMPLICIT NONE
….
CALL sub1( a, b, c, x, error)
….
END PROGRAM  main_prog
```

FORTRAN FUNCTIONS

A FORTRAN function is a procedure whose result is a single number, logical value, character string, or array. Two types of FORTRAN functions are intrinsic functions and user-defined functions (or function subprograms)

General form of FORTRAN function is

```
FUNCTION name (argument_list)
  ….
(Declaration section must declare type of name)
…..
(Execution section)
….
name = expr
RETURN
END FUNCTION   [name]
```

The function must begin with FUNCTION statement and end with END FUNCTION statement. The name of the function may be up to 31 alphabetic, numeric, and underscore characters long, but the first letter must be alphabetic. The name must be specified in the FUNCTION statement and is optional on the END FUNCTION statement.

A function is invoked by naming it in an expression. When a function is invoked, execution begins at the top of the function and ends when either a RETURN statement or the END FUNCTION statement is reached. RETURN statement is rarely used. If IMPLICIT NONE is used, the type of the function must be declared both in the function procedure and in the calling programs. If IMPLICIT NONE is not used, the default type of the function will follow the standard rules of FORTRAN unless they are overridden by a type declaration statement. The type declaration of a user defined FORTRAN function can take one of two equivalent forms:

```
INTEGER FUNCTION my_function ( i, j )
```

Or

```
FUNCTION my_function ( i, j )
INTEGER :: my_function
```

Be sure to declare the type of any user-defined functions both in the function itself and in any routines that call the function.