Scope in Fortran 90

The scope of objects (variables, named constants, subprograms) within a program is the portion of the program in which the object is visible (can be use and, if it is a variable, modified).

It is important to understand the scope of objects not only so that we know where to define an object we wish to use, but also what portion of a program unit is effected when, for example, a variable is changed, and, what errors might occur when using identifiers declared in other program sections.

Objects declared in a program unit (a main program section, module, or external subprogram) are visible throughout that program unit, including any internal subprograms it hosts. Such objects are said to be *global*. Objects are not visible between program units. This is illustrated in Figure 1.

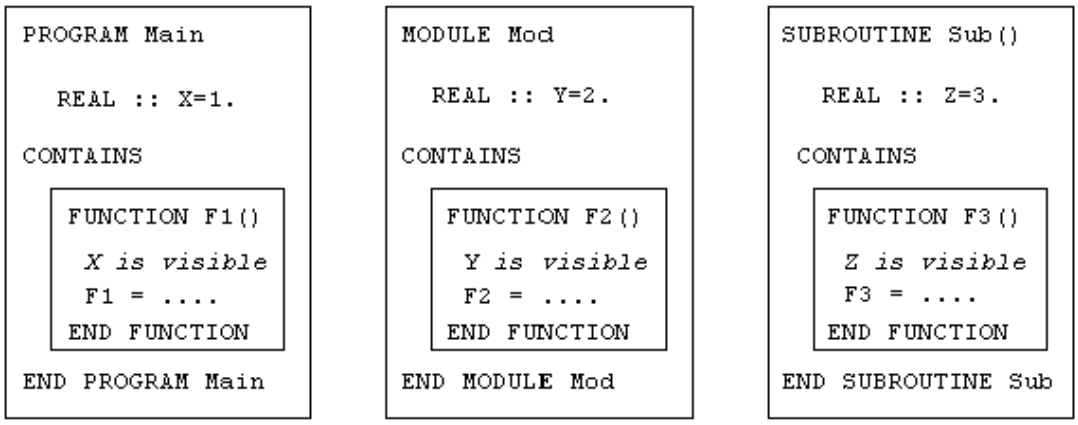

Figure 1: The figure shows three program units. Main program unit Main *is a host to the internal function* F1*. The module program unit* Mod *is a host to internal function* F2*. The external subroutine* Sub *hosts internal function* F3*. Objects declared inside a program unit are global; they are visible anywhere in the program unit including in any internal subprograms that it hosts. Objects in one program unit are not visible in another program unit, for example variable* X *and function* F3 *are not visible to the module program unit* Mod*.*

Objects in the module Mod can be imported to the main program section via the USE statement, see later in this section.

Data declared in an internal subprogram is only visible to that subprogram; i.e. it is *local*. If the data's name is already defined in the host then the host's data of the same name is no longer visible to the subprogram. It is good programming practice to declare in a subprogram all data that it uses - thus making them *local*. This is illustrated in Figure 2.

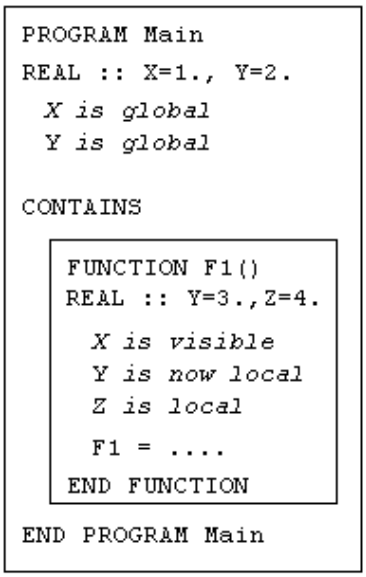

Figure 2: The figure shows a main program unit that hosts an internal function F1*. Variables* X *and* Y*, declared in the host, are global and so variable* X *is visible in the function. However, the function also declares* Y *and so* Y *becomes local and so its global value is no longer visible in the function.*

If all data is *local* in the subprogram then data must be communicated via arguments. Arguments are passed into functions, and, in and out of subroutines. Whether an argument is intended to only pass in or only passing out of the subprogram can be stated with the INTENT attribute. This is illustrated in Figure 3.

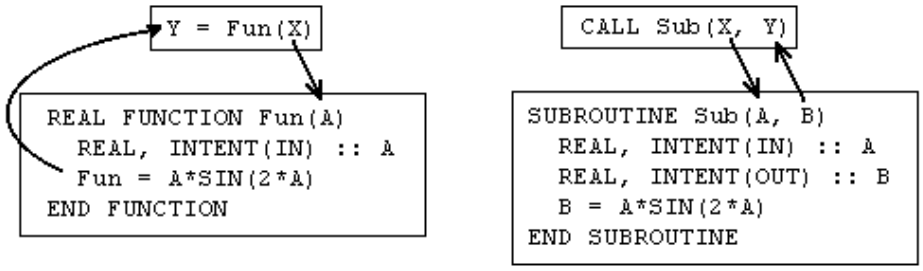

Figure 3: Data communicated in the form of arguments are given an INTENT *attribute. In the function reference* Y = Fun(X)*, actual argument* X *is passed to the formal argument* A *which is given the* INTENT(IN) *attribute; it therefore cannot be changed by the function. In the subroutine call* CALL Sub(X, Y)*, formal argument* A, INTENT(IN), *takes the value from actual argument* X *and cannot be modified in the subroutine, whereas, actual argument* Y *takes the result of the assignment of formal argument* B, INTENT(OUT)*.*

The INTENT attribute is optional, but when used it helps prevent unintentional use of arguments. One must, however, still be careful when writing names of identifiers inside a subprogram; if an identifier is misspelled but it is still valid because it is global, then the program will run *without* a compile- or run-time error.

In the case of a program unit that imports objects from a *module* via the USE statement, the scope of objects in the module then extends to that program unit. This is only a one way process, i.e objects in the program unit with the USE statement are not extended to the module it imports. This is illustrated in Figure 4.

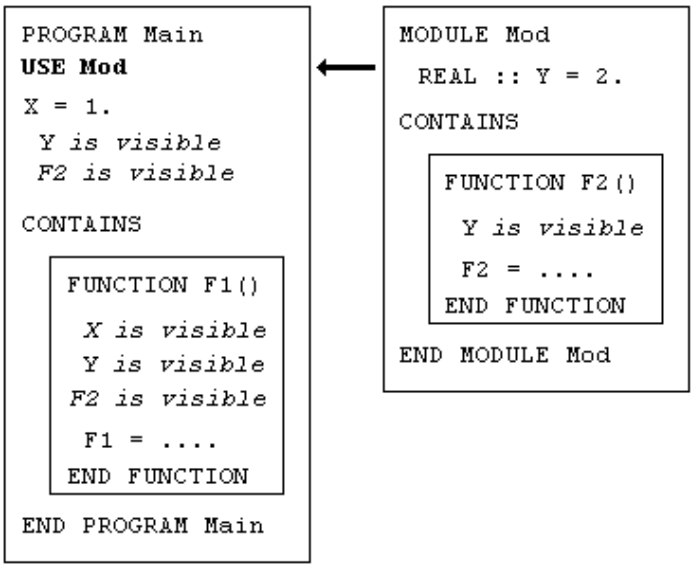

Figure 4: The main program unit imports objects from the module Mod *via the* USE *statement.* Y *and* F2 *become global objects in the main program unit.*

The programmer needs to understand scoping well enough to avoid problems that may occur due to scoping issues. For example a global variable may be modified in a subprogram - the side effect being that the modification is also seen in the whole program unit; if this is not the intent then there will probably be an undesirable effect on the program. If the variable is local to the subprogram then a modification to it in the subprogram is not visible to the rest of the program unit (and in other program units); if the intention is for the modification to be visible elsewhere then again there is a problem.

Tips:

Understanding comes with experience; experiment with subprograms and modules defining data in various sections and examine their scope.

For example:

```
PROGRAM Main                        PROGRAM Main
  REAL :: X=100. , Y=30.              REAL :: X=100. , Y=30.
  PRINT *, F1(), X, Y                 PRINT *, F1(), X, Y
CONTAINS                            CONTAINS
  FUNCTION F1()                       FUNCTION F1()
    Y = 55.                             REAL :: Y = 55.
    F1 = X*Y                            F1 = X*Y
  END FUNCTION                        END FUNCTION
END PROGRAM Main                    END PROGRAM Main

gives: 5500.0  100.0  55.0          gives: 5500.0  100.0  30.0
i.e. X and Y are global             i.e. X is global, but Y is local.
```

Modules are a good (organised) way to share global data between program units. They are also a safer way to provide subprograms as global variables in a main program unit are not visible in the module that hosts the subprograms.

Always use IMPLICIT NONE, and in subprograms declare all variables it uses thus making them local. State your intent when using arguments, i.e declare them and give them an INTENT attribute.

Large projects can be written in a highly modular form using modules. Unlike internal subprograms, a module subprogram can its self contain a subprogram. This is illustrated in Figure 5 below; function Cube is local to the host function Sphere and so is not

avialable to other subprograms or program units.

```
MODULE Volume

 REAL, PARAMETER :: PI=3.14159265

CONTAINS

  FUNCTION Sphere(R)
  REAL, INTENT(IN) :: R
  Sphere = 4./3.*PI*Cube(R)

  CONTAINS
   ┌─────────────────┐
   │FUNCTION Cube(A) │
   │  REAL, INTENT(IN) :: A
   │  Cube=A*A*A     │
   │END FUNCTION Cube│
   └─────────────────┘
  END FUNCTION Sphere

  SUBROUTINE Result(S)
    REAL,INTENT(IN) :: S
    PRINT *, "The result is ", S
  END SUBROUTINE Result

END MODULE Volume
```

Figure 5: Modular programming using modules. Module subprograms can themselves contain subprograms.