© UNIVERSAL RESEARCH REPORTS | REFEREED | PEER REVIEWED
ISSN: 2348 - 5612 | Volume: 04 Issue: 03 | July - September 2017

STUDY OF DERIVED PARAMETERS AND THE FORTRAN 90 DERIVED DATA TYPE

Apoorva Sharma, email: apoorva181092@gmail.com

ISSN : 2348-5612 © URR
9 770234 856124

Abstract : In many applications the intrinsic data types are not enough to express in code the ideas behind an algorithm or solution to a specific problem. Derived data types and structures allow programmer to group different kinds of information that belong to a single entity. In a way they resemble arrays but with two important differences. First the different elements of a derived data type do not have to be of the same type, thus they may include integer, character or real. Second the different entities making up the derived data type are referred to with a name and not an integer index. The different element of a derived data type are referred to as components. The data type of the component can be any of the intrinsic data types, or a previously defined derived data type.

The Derived Types :

A derived-type definition specifies a name for the type; this name is used to declare objects of the type. A derived-type definition also specifies components of the type, of which there must be at least one. A component can be of intrinsic or derived type; if it is of derived type, it can be resolved into components, called ultimate components. These ultimate components are of intrinsic type and can be pointers.

If the type definition appears in a module, the type definition may contain the keywords PUBLIC or PRIVATE. Generally, entities specified in a module can be kept private to the module and will not be available outside the module. This is true of data objects, module subprograms, and type definitions. By default, entities specified in a module are available to any program unit that accesses the module; that is, they have PUBLIC accessibility by default. This default can be changed by inserting a PRIVATE statement ahead of the specifications and definitions in the module. Individual entities can be specified to have either the PUBLIC or PRIVATE attribute regardless of the default. For a type definition, this can be accomplished by a PUBLIC or PRIVATE specifier in the TYPEstatement of the type definition.

© UNIVERSAL RESEARCH REPORTS | REFEREED | PEER REVIEWED
ISSN: 2348 - 5612 | Volume: 04 Issue: 03 | July - September 2017

The keyword PRIVATE can be used in two ways in type definitions in a module. One way makes the entire type private to the module; the other way allows the type name to be known outside the module, but not the names or attributes of its components. A separate PRIVATE statement that mentions the type name or a PRIVATE specifier in the TYPEstatement of the type definition provides the first of these.

An optional PRIVATE statement inside the type definition provides the second type definition can contain a SEQUENCE statement. The Fortran 90 standard allows a processor to rearrange the components of a derived type in any convenient order. However, if a SEQUENCE statement appears inside the type definition, the type is considered to be a sequence type. In this case, the processor must allocate storage for the components in the declared order so that structures declared to be of the derived type can appear in COMMON and EQUIVALENCE statements.

A derived type has a set of values that is every combination of the permitted values for the components of the type. The language provides a syntax for constants of the intrinsic types; it provides a somewhat similar mechanism, called a structure constructor, to specify a value for a derived type. A constructor can be used in the following places:

- In PARAMETER statements and in type declaration statements to define derived-type named constants
- In DATA statements to specify initial values
- As structure-valued operands in expressions

User-defined functions and subroutines must be used to define operations on entities of derived type. Thus, the four properties of the intrinsic types (possession of a name, a set of values, a set of operations, and a syntactic mechanism to specify constant values) are also provided for derived types.

Derived Type Definition

A derived type definition gives a derived type a name and specifies the types and attributes of its components. A derived type definition begins with a TYPE statement, ends with an END TYPE

© UNIVERSAL RESEARCH REPORTS | REFEREED | PEER REVIEWED
ISSN: 2348 - 5612 | Volume: 04 Issue: 03 | July - September 2017

statement, and has component declarations in between. The following example defines type PATIENT:

```
TYPE PATIENT
  INTEGER        PULSE_RATE
  REAL           TEMPERATURE
  CHARACTER*300  PROGNOSIS
END TYPE PATIENT
```

The format of a derived_type_def is as follows:

```
TYPE [ [, access_spec ] :: ] type_name
[ private_sequence_stmt ] ...
component_def_stmt
[ component_def_stmt ] ...
END TYPE [ type_name ]
```

Fortran 90: Derived Data Types

The Fortran 90 derived data type is similar to C structures and also has some similarities with C++ classes. The syntax for declaring a derived type, is

```
type mytype
  integer:: i
  real*8 :: a(3)
end type mytype
```

To create a variable of type mytype, use

```
type (mytype) var
```

An array of mytype can also be created.

```
type (mytype) stuff(3)
```

Elements of derived types are accessed with the "%" operator. For instance,

```
var%i = 3
var%a(1) = 4.0d0
```

© UNIVERSAL RESEARCH REPORTS | REFEREED | PEER REVIEWED
ISSN: 2348 - 5612 | Volume: 04 Issue: 03 | July - September 2017

stuff(1)%a(2) = 8.0d0

The real power of derived types comes with the ability to choose between functions (or subroutines) based on the type of their arguments. Different functions can be called by the same name, depending on whether the argument type is real, integer, or even a derived type. Intrinsic Fortran routines have always had this ability, the simplest example being choosing between single and double precision versions of the function. Now it can be extended to user's routines and defined data types as well. See the example in the section on intefaces for subroutines .

The compiler is free to store the constitutients of a derived type how it chooses. To force the derived type to be stored contiguously, use the sequence keyword. For example,

```
type mytype
  sequence
  integer:: i
  real*8 :: a(3)
end type mytype
```

There are many practical uses for derived data types. And, these data types function just like any other variable in a program unit, namely, they may be initialized, modified, and even passed to subprograms as parameters.

References :

1. https://www.rsmas.miami.edu/users/miskandarani/Courses/MSC321/lecttypes.pdf
2. http://csweb.cs.wfu.edu/~torgerse/Kokua/More_SGI/007-3692-006/sgi_html/ch04.html
3. Fortran Language Reference Manual, Volume 1
4. Fortran 90: Derived Data Types
5. https://courses.physics.illinois.edu/phys466/sp2013/comp_info/derived.html
6. https://web.stanford.edu/class/me200c/tutorial_90/10_derived.html