Fortran allows you to define derived data types. A derived data type is also called a structure, and it can consist of data objects of different types.

Derived data types are used to represent a record. E.g. you want to keep track of your books in a library, you might want to track the following attributes about each book:

- Title
- Author
- Subject
- Book ID

**Defining a Derived data type**

To define a derived data type, the type and end type statements are used. The type statement defines a new data type, with more than one member for your program. The format of the type statement is this:

```fortran
type type_name
  declarations
end type
```

Here is the way you would declare the Book structure:

```fortran
type Books
  character(len=50) :: title
  character(len=50) :: author
  character(len=150) :: subject
  integer :: book_id
end type Books
```

**Accessing Structure Members**

An object of a derived data type is called a structure.

A structure of type Books can be created in a type declaration statement like:

```fortran
type(Books) :: book1
```

The components of the structure can be accessed using the component selector character:

```fortran```
book1%title = "C Programming"
book1%author = "Nuha Ali"
book1%subject = "C Programming Tutorial"
book1%book_id = 6495407
```fortran```

**Note that there are no spaces before and after the % symbol.**

**Example**

The following program illustrates the above concepts:

```fortran
program deriveDataType
  ! type declaration
  type Books
    character(len=50) :: title
    character(len=50) :: author
```
character(len=150) :: subject
type :: book_id
end type Books

declaring type variables
type(Books) :: book1
type(Books) :: book2

accessing the components of the structure
book1%title = "C Programming"
book1%author = "Nuha Ali"
book1%subject = "C Programming Tutorial"
book1%book_id = 6495407

book2%title = "Telecom Billing"
book2%author = "Zara Ali"
book2%subject = "Telecom Billing Tutorial"
book2%book_id = 6495700

display book info

Print *, book1%title
Print *, book1%author
Print *, book1%subject
Print *, book1%book_id

Print *, book2%title
Print *, book2%author
Print *, book2%subject
Print *, book2%book_id

end program deriveDataType

When the above code is compiled and executed, it produces the following result:

C Programming
Nuha Ali
C Programming Tutorial
6495407
Telecom Billing
Zara Ali
Telecom Billing Tutorial
6495700

Array of Structures

You can also create arrays of a derived type:

type(Books), dimension(2) :: list

Individual elements of the array could be accessed as:

list(1)%title = "C Programming"
list(1)%author = "Nuha Ali"
list(1)%subject = "C Programming Tutorial"
list(1)%book_id = 6495407

The following program illustrates the concept:

program deriveDataType

! type declaration


type Books
character(len=50) :: title
character(len=50) :: author
character(len=150) :: subject
integer :: book_id
end type Books

! declaring array of books
type(Books), dimension(2) :: list

! accessing the components of the structure

list(1)%title = "C Programming"
list(1)%author = "Nuha Ali"
list(1)%subject = "C Programming Tutorial"
list(1)%book_id = 6495407

list(2)%title = "Telecom Billing"
list(2)%author = "Zara Ali"
list(2)%subject = "Telecom Billing Tutorial"
list(2)%book_id = 6495700

! display book info
Print *, list(1)%title
Print *, list(1)%author
Print *, list(1)%subject
Print *, list(1)%book_id

Print *, list(2)%title
Print *, list(2)%author
Print *, list(2)%subject
Print *, list(2)%book_id

end program deriveDataType

When the above code is compiled and executed, it produces the following result:

C Programming
Nuha Ali
C Programming Tutorial
  6495407
C Programming
Zara Ali
Telecom Billing Tutorial
  6495700

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js