20

User Defined Types

"Russell's theory of types leads to certain complexities in the foundations of mathematics... Its interesting features for our purposes are that types are used to prevent certain erroneous expressions from being used in logical and mathematical formulae; and that a check against violation of type constraints can be made purely by scanning the text, without any knowledge of the value which a particular symbol might happen to have."

C.A.R. Hoare, *Structured Programming*

Aims

The aim of this chapter is to introduce the concepts and ideas involved in using the facilities offered in Fortran 90 for the construction and use of user defined types:

- The way in which we define our own types.
- The way in which we declare variables to be of a user defined type.
- The way in which we manipulate variables of our own types.
- The way in which we can nest types within types.

The examples are simple and are designed to highlight the syntax. More complex and realistic examples of the use of user defined data types are to be found in later chapters.

20 User Defined Types

In the coverage so far we have used the intrinsic types provided by Fortran. The only data structuring technique available has been to construct arrays of these intrinsic types. Whilst this enables us to solve a reasonable variety of problems, it is inadequate for many purposes. In this chapter we look at the facilities offered by Fortran for the construction of our own types and how we manipulate data of these new, user defined types.

With the ability to define our own types we can now construct aggregate data types that have components of a variety of base types. These are often given the name records in books on data structures. In mathematics the term cartesian product is often used, and this is the terminology adopted by Hoare. We will stick to the term records, as it is the one that is most commonly used in computing and texts on programming.

There are two stages in the process of creating and using our own data types: we must first define the type, and then create variables of this type.

20.1 Example 1: Dates

```
PROGRAM ch2001
IMPLICIT NONE
TYPE Date
   INTEGER :: Day=1
   INTEGER :: Month=1
   INTEGER :: Year=2000
END TYPE Date
TYPE (Date) :: D
   PRINT *,D%Day, D%Month, D%Year
   PRINT *,' Type in the date, day, month, year'
   READ *,D%Day, D%Month, D%Year
   PRINT *,D%Day, D%Month, D%Year
END PROGRAM ch2001
```

This complete program illustrates both the definition and use of the type. It also shows how you can define initial values within the type definition.

20.2 Type definition

The type *Date* is defined to have three component parts, comprising a *day*, a *month* and a *year*, all of integer type. The syntax of a type construction comprises:

TYPE Typename
  Data Type :: Component_name
  etc
END TYPE Typename

Reference can then be made to this new type by the use of a single word, *Date*, and we have a very powerful example of the use of abstraction.

20.3 Variable definition

This is done by

```
TYPE (Typename) :: Variablename
```

and we then define a variable *D* to be of this new type. The next thing we do is have a READ * statement that prompts the user to type in three integer values, and the data are then echoed straight back to the user. We use the notation `Variablename%Component_Name` to refer to each component of the new data type.

20.4 Example 2: Address lists

```
PROGRAM ch2002
IMPLICIT NONE
TYPE Address
   CHARACTER (LEN=40) :: Name
   CHARACTER (LEN=60) :: Street
   CHARACTER (LEN=60) :: District
   CHARACTER (LEN=60) :: City
   CHARACTER (LEN=8)  :: Post_Code
END TYPE Address
INTEGER , PARAMETER :: N_of_Address=78
TYPE (Address) , DIMENSION(N_of_Address):: Addr
INTEGER :: I
   OPEN(UNIT=1,FILE="ADDRESS.DAT")
   DO I=1,N_of_Address
      READ(UNIT=1,FMT='(A40)') Addr(I)%Name
      READ(UNIT=1,FMT='(A60)') Addr(I)%Street
      READ(UNIT=1,FMT='(A60)') Addr(I)%District
      READ(UNIT=1,FMT='(A60)') Addr(I)%City
      READ(UNIT=1,FMT='(A8)') Addr(I)%Post_Code
   END DO
   DO I=1,N_of_Address
      PRINT *,Addr(I)%Name
```

```
      PRINT *,Addr(I)%Street
      PRINT *,Addr(I)%District
      PRINT *,Addr(I)%City
      PRINT *,Addr(I)%Post_Code
   END DO
END PROGRAM ch2002
```

In this example we define a type Address which has components that one would expect for a person's address. We then define an array Addr of this type. Thus we are now creating arrays of our own user defined types. We index into the array in the way we would expect from our experience with integer, real and character arrays. The complete example is rather trivial in a sense in that the program merely reads from one file and prints the file out to the screen. However, it highlights many of the important ideas of the definition and use of user defined types.

20.5 Example 3: Nested user defined types

The following example builds on the two data types already introduced. Here we construct nested user defined data types based on them and construct a new data type containing them both plus additional information:

```
PROGRAM ch2003
IMPLICIT NONE
TYPE Address
   CHARACTER (LEN=60)   :: Street
   CHARACTER (LEN=60)   :: District
   CHARACTER (LEN=60)   :: City
   CHARACTER (LEN=8 )   :: Post_Code
END TYPE Address
TYPE Date_Of_Birth
   INTEGER :: Day
   INTEGER :: Month
   INTEGER :: Year
END TYPE Date_Of_Birth
TYPE Personal
   CHARACTER (LEN=20)    :: First_Name
   CHARACTER (LEN=20)    :: Other_Names
   CHARACTER (LEN=40)    :: Surname
   TYPE (Date_Of_Birth) :: DOB
   CHARACTER (LEN=1)     :: Sex
   TYPE (Address)         :: Addr
END TYPE Personal
INTEGER , PARAMETER :: N_People=2
```

```
TYPE (Personal) , DIMENSION(N_People) :: P
INTEGER :: I
   OPEN(UNIT=1,FILE='PERSON.DAT')
   DO I=1,N_People
      READ(1,FMT=10) P(I)%First_Name,&
                     P(I)%Other_Names,&
                     P(I)%Surname,&
                     P(I)%DOB%Day,&
                     P(I)%DOB%Month,&
                     P(I)%DOB%Year,&
                     P(I)%Sex,&
                     P(I)%Addr%Street,&
                     P(I)%Addr%District,&
                     P(I)%Addr%City,&
      P(I)%Addr%Post_Code
      10 FORMAT(  A20,/,&
                  A20,/,&
                  A40,/,&
                  I2,1X,I2,1X,I4,/,&
                  A1,/,&
                  A60,/,&
                  A60,/,&
                  A60,/,&
                  A8)
   END DO
   DO I=1,N_People
      WRITE(*,FMT=20)   P(I)%First_Name,&
                        P(I)%Other_Names,&
                        P(I)%Surname,&
                        P(I)%DOB%Day,&
                        P(I)%DOB%Month,&
                        P(I)%DOB%Year,&
                        P(I)%Sex,&
                        P(I)%Addr%Street,&
                        P(I)%Addr%District,&
                        P(I)%Addr%City,&
                        P(I)%Addr%Post_Code
      20 FORMAT(        A20,A20,A40,/,&
                        I2,1X,I2,1X,I4,/,&
                        A1,/,&
                        A60,/,&
                        A60,/,&
```

```
                              A60,/,&
                              A8)
   END DO
END PROGRAM ch2003
```

Here we have a date of birth data type (`Date_Of_Birth`) based on the `Date` data type from the first example, plus a slightly modified address data type, incorporated into a new data type comprising personal details. Note the way in which we reference the component parts of this new, aggregate data type.

20.6 Problems

1. Modify the last example to include a more elegant printed name. The current example will pad with blanks the first name, other names and surname and span 80 characters on one line, which looks rather ugly.

Add a new variable name which will comprise all three subcomponents and write out this new variable, instead of the three subcomponents.

20.7 Bibliography

Dahl O.J., Dijkstra E.W., Hoare C.A.R., *Structured Programming*, Academic Press, 1972.

- This is one of the earliest and best introductions to data structures and structured programming. The whole book hangs together very well, and the section on data structures is a must for serious programmers.

Vowels R.A., *Algorithms and Data Structures in F and Fortran*, Unicomp, 1989.

- One of the few books looking at algorithms and data structures using Fortran.

Wirth N., *Algorithms + Data Structures = Programs*, Prentice-Hall, 1976.

Wirth N., *Algorithms + Data Structures*, Prentice-Hall, 1986.

- The first is in Pascal, and the second in Modula 2.

Wood D., *Paradigms and Programming in Pascal*, Computer Science Press, 1984.

- Contains a number of examples of the use of recursion in problem solving. Also provides a number of useful case studies in problem solving.