Structures in QBasic

(Custom *data-types* using the **TYPE** statement)

A complete **TYPE** *statement* in *QBasic* describes a fixed-length *record* (custom *data-type*). With a **TYPE** *statement*, we describe our *record(s)* (custom *data-types*) that can contain any mixture of *fields* with any combination of *data-types*. The format of **TYPE** is:

```
TYPE RECORDNAME
     fieldname as datatype
     [fieldname2 as data type1]
     .
     .
     .
     [fieldnameN as datatypeN]
END TYPE
```

We can think of our complete **TYPE** *statement* as a *record* (custom *data-type*) description. Our **RECORDNAME**(s) can consist of 1 to 40 letters and numbers but cannot include any special characters, such as a period or an underscore character.

To begin creating a *record* (custom *data-type*) that contains CD titles we have in our personal CD collection, for example, we'll need to start with a **TYPE** statement. As in the following:

```
TYPE MUSICREC
```

Our **RECORDNAME** (**MUSICREC**) has no *data-type* suffix because it's a totally new *data-type*, one that we are creating. **RECORDNAME** (**MUSICREC**) is the name we will refer to when we want to *instantiate* (create) ***variables/objects*** that look like the named *record* (**MUSICREC**). The rest of the **TYPE** *statement* describes each *field* in the *record*. **fieldname** can be any name we specify (also consisting of 1 to 40 letters and numbers), and **datatype** is any QBasic-defined/intrinsic *data-type* listed below.

The TYPE statement's possible field data types and their lengths.		
Data Type	***Description***	***Length In bytes***
INTEGER	Integer	2
LONG	Long integer	4
SINGLE	Single-precision	4
DOUBLE	Double-precision	8
STRING * *N*	Fixed-length string	*N*

To describe the rest of our CD collection in our example, we can use the following (complete) **TYPE** statement:

```
TYPE MUSICREC
     title AS STRING * 20      'Title of the CD
     quantity AS INTEGER       'Number of the CD I have
     condition AS STRING * 5   'GOOD, POOR, etc.
     numSongs AS INTEGER       '# of songs on the CD
     pricePd AS SINGLE         'Price I paid for CD
END TYPE
```

NOTE: Each *field* we define that contains a *string data-type* must also include a fixed *string length* (**title AS STRING * 20**).

By writing our complete **TYPE** statement, we prepare our program so that it can read and write *records* that look like the one we described.

A complete **TYPE** statement only describes our *records*; it does not reserve any storage/memory. In order to reserve memory we must define one or more *record* (**MUSICREC**) ***variable(s)*** of the custom *data-type* described by our **TYPE** statement. Because programs store data in ***variables***, we need to create *record* variables (in the same way we have been creating *integer*, *long-integer*, *string*, *single-precision*, and *double-precision* ***variables***).

The following line is used to create three **cd *variables*** (**cd1, cd2, cd3**) for our compact-disc *record* (custom *data-type*)**:**

```
DIM cd1 AS MUSICREC, cd2 AS MUSICREC, cd3 AS MUSICREC
```

This line creates three ***variables***; **cd1**, **cd2**, and **cd3 e**ach of which has its *type* defined in our **TYPE** statement; in other words, **cd1** consists of:

- a 20-character *string*
- followed by an *integer*,
- followed by a five-character *string*,
- followed by an *integer*
- followed by a *single-precision* value

as do **cd2**, and **cd3**.

With the following line, we can create an *array* of *record* ***variables***:

```
DIM cds(1 to 500) as MUSICREC
```

The *type* of the new *array* is not going to be like our previous *string* or *integer data-type arrays.* Our *array type* is defined as our custom *data-type* - **MUSICREC**. Each element in our *array* looks like the *record* (custom *data-type*) we defined in our **TYPE MUSICREC** statement.

The following program snippet defines only one *record* ***variable***, **cd**; *title*, *quantity*, *condition*, *numsongs*, and *pricepd* are not ***variables***, but **fieldnames** for the *record* **MUSICREC**. After we create the **TYPE MUSICREC** custom *data-type* and *instantiate* (create) the **cd** ***variable*** in memory…

```
TYPE MUSICREC
     title AS STRING * 20      'Title of the CD
     quantity AS INTEGER       'Number of the CD I have
     condition AS STRING * 5   'GOOD, POOR, etc.
     numSongs AS INTEGER       '# of songs on the CD
     pricePd AS SINGLE         'Price I paid for CD
END TYPE

DIM cd AS MUSICREC             'Creates one record variable
```

…we still need to put values in it.

Our ***variable*** **cd** refers to a single ***variable*** in memory. In order to fill this *record* ***variable*** (**cd**), we're going to need to use what is called the **dot** operator (**.**). Following is an example:

```
cd.title = "Bruno's Here Again!"
cd.quantity = 1
cd.condition = "Good"
cd.numSongs = 12
cd.pricePd = 9.75
```

Note**:** to assign values to *fields* in a *record* (custom *data-type*) ***variable***, we need to type a specific **RECORDNAME** (**cd**), followed by the **dot** operator (a period); For this reason, *record* names and *field* names cannot contain a period. Using the **dot** operator puts values in the *record(s)* fields.

RECORDNAME.fieldname are used in pairs to create and access individual **fieldname** values which we can *print*, *assign*, or *pass* to *subroutines* and *functions*. In the pair, **cd.title, cd** specifies the *record* (custom *data-type*) ***variable*** by name, and **title** specifies the **fieldname** in that *record*. An assignment to this pair could look like this:

```
cd.title = "Bruno's Here Again!"
```

SCOPE (for future reference):

The following three ***variables***…

```
DIM cd1 AS MUSICREC, cd2 AS MUSICREC, cd3 AS MUSICREC
```

are local to the *routine/subroutine/function* that creates them. If we want to make them *global*, we need to use the **COMMON SHARED** *statement* in the main program as follows:

```
COMMON SHARED cd1 AS MUSICREC, cd2 AS MUSICREC, cd3 AS MUSICREC
```

If we use the **SHARED** statement (without the **COMMON** *keyword*) it makes the ***variables*** *local* to the *routine(s)* that contain the same **SHARED** *statement*, as in this line:

```
SHARED cd1 AS MUSICREC, cd2 AS MUSICREC, cd3 AS MUSICREC
```

If we pass a *record* that has been created with the **TYPE** statement to a *subroutine*/*function*, the receiving *subroutine/function* receives it with an **AS ANY** (**TYPE**) in its *parameter* list.

The following *subroutine*, for example receives a *record* called **custrec** that was passed to it by a *subroutine call* in another part of the program:

```
DisplayCust(custrec AS ANY)
```
