ARRAYS IN BASIC

TABLE OF CONTENTS

1. Arrays as Variables p.1

2. Examples of Array Usage p.6

This document was written by Ken Karp for posting on the QuickBASIC conference of MicroSellar BBS. Thanks to Terri Karp for valuable assistance in editing. For more information about BASIC programming, and QuickBASIC in particular, register with the MicroSellar BBS at (201) 239-1346 (modem). Registration is free. Participation is welcome.

ARRAYS IN BASIC

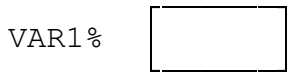
1. ARRAYS AS VARIABLES

When a BASIC program is running, the first time a reference is made to a particular variable, a `box' is created for that variable in the computer's memory.

Examine the following BASIC program segment:

```
                                        '  / Input a value from the
    INPUT VAR1%                         ' <  keyboard and set VAR1
                                        '  \ equal to it
    VAR1% = VAR1% + 10                  'manipulate the variable
    PRINT "The value of VAR1 is";VAR1%  'output the value
```

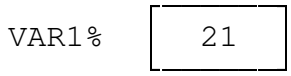
Let's focus in on the first statement. If this is the first reference to VAR1%, the computer will set aside a box for it:



When we think of the value of VAR1%, we can think of whatever is inside the box. Thus, when the BASIC statement

```
                                        '  / Input a value from the
    INPUT VAR1%                         ' <  keyboard and set VAR1
                                        '  \ equal to it
```

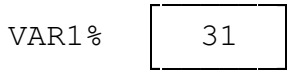
is encountered, a check is made to see if a box had previously been created for VAR1%. If no box is found, one will be created, and only after that is done will the "INPUT" part of the statement be performed. If the user enters the value 21 to the INPUT prompt, we will have the following:



In light of all this, our second statement takes on the following meaning: "go get the value that is stored in the box for VAR1%; add 10 to it; store the result back in the box for VAR1%". Thus, after the statement

```
    VAR1% = VAR1% + 10                  'manipulate the variable
```

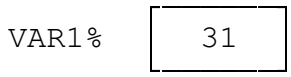
has executed, we have:



Finally, the third statement

```
PRINT "The value of VAR1 is";VAR1%    'output the value
```

will get the value that is stored in the box for VAR1%; and then print it. Although VAR1% box is referenced by this statement, its contents remain unchanged, and as our program segment finishes, we still have:



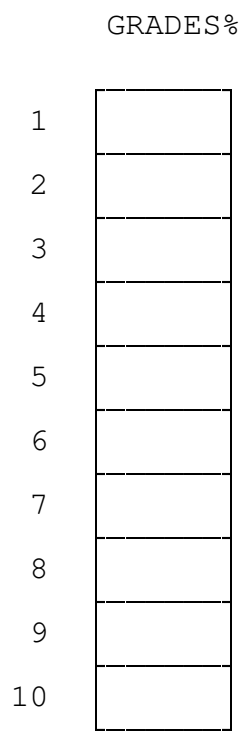
Pursuing this analogy, an array is nothing more than a whole bunch of the same type of boxes. The BASIC DIM statement is used to announce that a certain number of boxes are being set aside, AND that they are to be referred to by a certain name. Thus, the BASIC statement

```
DIM GRADES%(10)
```

will set aside 10 [*] integer boxes each of which are to be referred to using the name GRADES%. Each of the 10 boxes is capable of holding a value, just like VAR1% in the example above. In effect, we can imagine the boxes to look something like this:

---

* - Actually, the statement will set aside 11 boxes, numbered 0 through 10; but for purposes of this discussion, it is simpler to assume 10 boxes. QuickBASIC users, look up OPTION BASE in your manual.



But how (the attentive student will ask) are the 10 boxes distinguished from each other if they are all called GRADES%? Simple: by putting the number of the box desired in parentheses immediately following the word GRADES%. Thus,

```
FIRSTGRADE% = GRADES%(1)
```

will find the first of the 10 boxes associated with the name GRADES%, extract the value currently found there, and place it into some other box, called FIRSTGRADE%.

```
GRADES%(4) = FOURTHONE%
```

will copy the value currently in the box named FOURTHONE% into the fourth box associated with the variable name GRADES%.

In BASIC, the number inside the parentheses is called the "subscript", and arrays are often called "subscripted variables". It is important to note that the subscript itself may be a variable. Observe the following program segment:

```
PRINT "Here are the class's grades:"    'print an introduction
FOR N%=1 TO 10                         'do something 10 times
    PRINT GRADES%(N%)             'print what's in the Nth box
NEXT
```

If we had:

GRADES%

1	94
2	97
3	83
4	91
5	79
6	82
7	92
8	88
9	100
10	89

prior to executing the loop, we would see results something like this on our screen:

```
Here are the class's grades:
 94
 97
 83
 91
 79
 82
 92
 88
 100
 89
```

Why? Since the variable N% is used as the counter in the FOR loop, each time the statement:

```
PRINT GRADES%(N%)            'print what's in the Ith box
```

is executed, N% has a value one greater than the last time it was executed; therefore, the SUBSCRIPT into GRADES% causes the program to look at the NEXT numbered box.

2. EXAMPLES OF ARRAY USAGE

There are three QuickBASIC examples of array usage distributed with this discussion. They are

ONE-D.BAS
TWO-D.BAS
THREE-D.BAS

ONE-D.BAS employs a one dimensional array called CITIES$(). A one dimensional array in BASIC uses only one subscript in all references to the array (in the above example GRADES%() is a one dimensional array). ONE-D.BAS reads in a list of city names from DATA statements and stores them in CITIES$(). It then prints out the same list of cities from CITIES$().

If a BASIC array has two subscripts it is referred to as a two dimensional array. An example of a two dimensional array is TEMPS(), found in TWO-D.BAS. The statement

```
DIM  ...., TEMPS(100,4)
```

sets aside storage for a 100x4 (actually 101x5 -- TEMPS(0,0) is valid and may be used if desired!!) array. Notice the FOR-NEXT loop in the subroutine READINDATA: each time the loop counter (I) varies (ie, with each iteration of the loop), we read in an entire row of data (ie, TEMPS(I,1), TEMPS(I,2), TEMPS(I,3), & TEMPS(I,4)). The same stragegy is used when the values are printed out.

The final example program, THREE-D.BAS, takes us one step further: now we have added a third dimension. TEMPS() uses its first dimension for the city; its second dimension for the particular data in question (ie, high temperature, low temperature, rainfall, or wind speed); and its third dimension for the day of the week (Sunday through Saturday). Just as a two dimensional array may be conceptualized as a table of values, a three dimensional array may be conceptualized as an accountant's ledger: when the ledger is open to Sunday, you see before you a table of figures (a two dimensional array!); when you turn the page to Monday, you see a similar table, but the actual figures may differ; when you turn to Tuesday, again the figures may differ. In actual fact, TEMPS() is a 4x100x6 structure: 2400 integers, all referenced by the name TEMPS().

The program THREE-D.BAS prints all the weather information for the 22 cities for a certain day of the week. By pressing <PgUp> or <PgDn>

the user can print the same information for a different day of the week. To do this, the subroutine PRINTOUTDATA holds the third subscript steady (variable DAY is set before PRINTOUTDATA is called) while varying the other two subscripts. This yields a two dimensional cross-section of our three dimensional array: ie, all the weather information for all the cities, but only for a particular day. In fact, that is the two dimensional array we see on the screen.

QuickBASIC users will note that v3.0 allows up to 63 dimensions for an array! That is to say

```
DIM ARRAYNAME (d1,d2,d3,d4, ... ,d63)
```

would set aside storage for d1xd2xd3xd4x ... xd63 array elements -- a very large data structure indeed! In actual practice, arrays of more than two or three dimensions are rarely used. It should be easy to see why.