Chapter 15: Arrays – Collections of Information.

We have used simple string and numeric variables in many programs, but they can only contain one value at a time. Often we need to work with collections or lists of values. We can do this with either one-dimensioned or two-dimensioned arrays. This chapter will show you how to create, initialize, use, and re-size arrays.

One-Dimensional Arrays of Numbers:

A one-dimensional array allows us to create a list in memory and to access the items in that list by a numeric address (called an index). Arrays can contain any type of value (integer, decimal, or string).

Our first example of an array will be using numeric values.

```
1     # arraynumeric1d.kbs
2     # one-dimensional numeric array
3
4     dim a(4)
5
6     a[0] = 100
7     a[1] = 200
8     a[2] = a[0] + a[1]
9
10    inputfloat "Enter a number> ", a[3]
11
12    for t = 0 to 3
13         print "a[" + t + "] = " + a[t]
14    next t
```

Program 89: One-dimensional Numeric Array

```
Enter a number> 63
a[0] = 100
a[1] = 200
a[2] = 300
a[3] = 63.0
```

Sample Output 89: One-dimensional Numeric Array


```
dim variable(items)
dim variable(rows, columns)
dim variable(items) fill expression
dim variable(rows, columns) fill expression
```

The **dim** statement creates an array in the computer's memory the size that was specified in the parenthesis. Sizes (items, rows, and columns) must be integer values greater than or equal to one (1).

The **dim** statement will NOT initialize the elements in the new array unless you specify a fill value. The **fill** clause will assign the value to all elements of the array.

***variable*[*index*]**
***variable*[*rowindex*, *columnindex*]**

You can use an array reference (variable with index(s) in square brackets) in your program almost anywhere you can use a simple variable. The index or indexes must be integer values between zero (0) and one less than the size used in the *dim* statement.

It may be confusing, but BASIC-256 uses zero (0) for the first element in an array and the last element has an index one less than the size. Computer people call this a zero-indexed array.

Arrays can also be used to store string values. All you have to do is store a string in the array element.

```
15    # listoffriends.kbs
16    # use an array to store any number of names
17
18    print "make a list of my friends"
19    inputinteger "how many friends do you have?", n
20
21    dim names(n)
22    for i = 0 to n-1
23         input "enter friend name ?", names[i]
24    next i
25
26    # show the names
27    cls
28    print "my friends"
29    for i = 0 to n-1
30         print "friend number ";
31         print i + 1;
32         print " is " + names[i]
33    next i
34
35    # pick one at random
```

```
36    x = int(rand * n)
37    print "The winner is " + names[x]
38    end
```

Program 90: List of My Friends

```
make a list of my friends
how many friends do you have?3
enter friend name ?Kendra
enter friend name ?Bob
enter friend name ?Susan
 - screen clears -
my friends
friend number 1 is Kendra
friend number 2 is Bob
friend number 3 is Susan
The winner is Kendra
```

Sample Output 90: List of My Friends

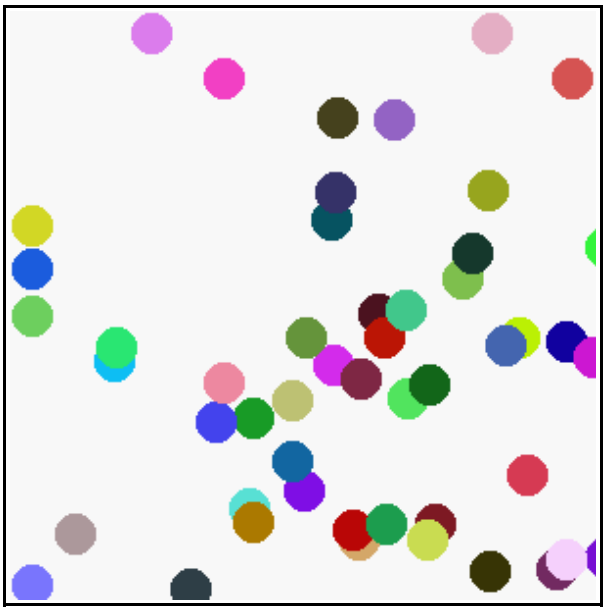
We can use arrays of numbers to draw many balls bouncing on the screen at once. Program 89 uses 5 arrays to store the location of each of the balls, it's direction, and color. Loops are then used to initialize the arrays and to animate the balls. This program also uses the **rgb()** function to calculate and save the color values for each of the balls.

```
1     # manyballbounce.kbs
2     # use arrays to keep up with the direction,
3     # location, and color of many balls on the screen
4
5     fastgraphics
6
7     r = 10  # size of ball
8     balls = 50  # number of balls
```

```
9
10    # position of the balls - start them all at 0,0
11    dim x(balls) fill 0
12    dim y(balls) fill 0
13
14    # speed of the balls (set randomly)
15    dim dx(balls)
16    dim dy(balls)
17
18    # color of the balls (set randomly)
19    dim colors(balls)
20
21    for b = 0 to balls-1
22         # speed in x and y directions
23         dx[b] = rand * r + 2
24         dy[b] = rand * r + 2
25         # each ball has it's own color
26         colors[b] = rgb(rand*256, rand*256, rand*256)
27    next b
28
29    color green
30    rect 0,0,300,300
31
32    while true
33         # erase screen
34         clg
35         # now position and draw the balls
36         for b = 0 to balls -1
37              # move ball to new location
38              x[b] = x[b] + dx[b]
39              y[b] = y[b] + dy[b]
40              # if off the edges turn the ball around
41              if x[b] < 0 or x[b] > graphwidth then
42                   dx[b] = dx[b] * -1
43              end if
44              # if off the top of bottom turn the ball
      around
45              if y[b] < 0 or y[b] > graphheight then
46                   dy[b] = dy[b] * -1
```

```
47              end if
48              # draw new ball
49              color colors[b]
50              circle x[b],y[b],r
51          next b
52          # update the display
53          refresh
54          pause .05
55      end while
```

Program 91: Bounce Many Balls



Sample Output 91: Bounce Many Balls

Assigning Arrays:

We have seen the use of the curly brackets ({}) to play music, draw polygons, and define stamps. The curly brackets can also be used to create and assign an entire array with custom values.

```
1     # arrayassign.kbs
2     # using a list of values to create an assign an array
3
4     numbers = {56, 99, 145}
5     names = {"Bob", "Jim", "Susan"}
6
7     for i = 0 to 2
8        print numbers[i] + " " + names[i]
9     next i
```