Multi-dimensional Arrays

In this lesson, we discuss multi-dimensional arrays.

WE'LL COVER THE FOLLOWING

Dynamic multi-dimensional arrays

Arrays of arrays are called *multi-dimensional arrays*. The elements of all of the arrays that we have defined so far have been written in the source code from left to right. To help us understand the concept of a two-dimensional array, let's define an array from top to bottom this time:

```
int[] array = [
               10,
               20,
               30,
               40
              ];
```

As you remember, most spaces in the source code are used to help with readability and do not change the meaning of the code. The array above could have been defined on a single line and would have the same meaning.
Let's now replace every element of that array with another array:

```
/* ... */ array = [
                   [ 10, 11, 12 ],
                   [ 20, 21, 22 ],
                   [ 30, 31, 32 ],
                   [ 40, 41, 42 ]
```

```
              ];
```

We have replaced elements of type `int` with elements of type `int[]`. To make the code conform to the array definition syntax, we must now specify the type of the elements as `int[][]` instead of `int[]`:

```
int[][] array = [
                  [ 10, 11, 12 ],
                  [ 20, 21, 22 ],
                  [ 30, 31, 32 ],
                  [ 40, 41, 42 ]
                ];
```

Such arrays are called **two-dimensional arrays** because they can be seen as having rows and columns.

Adding elements in a multidimensional array

Two-dimensional arrays are used the same way as any other array as long as we remember that each element is an array itself and can be used in array operations:

```
array ~= [ 50, 51 ]; // adds a new element (i.e. a slice)
array[0] ~= 13; // adds to the first element
```

The new state of the array:

```
[[10, 11, 12, 13], [20, 21, 22], [30, 31, 32], [40, 41, 42], [50, 51]]
```

```
import std.stdio;

void main() {
    int[][] array = [                // Here array is a dynamic array and can have
                     [ 10, 11, 12 ], // variable-length arrays
                     [ 20, 21, 22 ],
                     [ 30, 31, 32 ],
                     [ 40, 41, 42 ]
                   ];

    array ~= [ 50, 51 ]; // adds a new element (i.e. a slice)
    array[0] ~= 13; // adds to the first element

    writeln(array);

}
```

Fixed-length multidimensional arrays

Multidimensional arrays and their elements can be fixed-length as well The following is a three- dimensional array where all dimensions are fixed-length:

```
int[2][3][4] array; // 2 columns, 3 rows, 4 pages
```

The definition above can be seen as four pages of three rows of two columns of integers. As an example, such an array can be used to represent a 4-story building in an adventure game, where each story consists of 2x3=6 rooms. For example, the number of items in the first room of the second floor can be incremented like the following:

```
// The index of the second floor is 1, and the first room of that floor i
s accessed by [0][0]
++itemCounts[1][0][0];
```

```
import std.stdio;

void main() {
    int[2][3][4] itemCounts; // 2 columns, 3 rows, 4 pages

    ++itemCounts[1][0][0];

    writeln(itemCounts);
}
```

Slicing in multidimensional arrays

In addition to the syntax above, the new expression can also be used to create a slice of slices. The following example uses only two dimensions:

```
import std.stdio;

void main() {
    int[][] s = new int[][](2, 3);

    writeln(s);
}
```

Slice of slices

The new expression above creates 2 slices containing 3 elements each and returns a slice that provides access to those slices and elements.

In the next lesson, you will find a coding challenge related to arrays and slices.