Arrays

Like vectors, the array is a data structure used in C++ to store a sequential collection of elements. Unlike vectors, its size cannot be changed.

Being able to store multiple pieces of related information in the same structure is very useful when writing C++ programs. One way we can do that is by using vectors:

```
std::vector<int> favoriteNums = {7, 9, 15, 16};

std::cout << favoriteNums[2]; // Prints: 15
```

Vectors are a modern approach to handling information. But C++ also inherits a more low-level means of doing this from its parent language, C, called *arrays*.

Arrays in C++ are similar to vectors in that they allow us to store groups of information. However, arrays are ultimately lower-level constructs and require some more work on the part of the user.

Arrays vs. Vectors

If you've used C++ vectors in the past, you may be wondering what exactly the difference is between them and arrays, and when you should use which.

As was mentioned earlier, arrays are inherited from C, C++'s parent language. They are a low-level data structure and are incredibly rigid. *With arrays, you can't add or remove elements; you can only modify existing elements.*

Vectors are originated from arrays. Early in the creation of C++, the language developers took these basic arrays and wrote code to enhance them and make them more flexible and powerful. Therefore you can think of vectors as super arrays!

Vectors don't require a static size. It's possible to add and remove elements from them, as well as access their current size at any time.

Creating an Array

When creating an array, you have to keep two pieces of information in mind:

1. The type of data you want to store inside of it.
2. How many items you want it to be able to hold (its size).

We can create an array a lot like we create normal variables, by specifying the data type, giving it a descriptive name, and also specifying its size:

```
int favoriteNums[4];
```

In the above code example, we've created an array with a size of `4`, meaning it can hold four integers (all four elements will initially have the default `int` value of `0`).

In many cases, you won't know what data needs to go in the array until after you've created it, but if you do happen to know the contents of the array ahead of time, you can initialize it with custom values upfront:

```
int favoriteNums[] = {7, 9, 15, 16};
```

This array would also have a size of `4`, but we don't need to explicitly specify that when we initialize it in this way.

Array Indices

Like vectors, each element in an array is assigned a specific index starting at zero. To access or modify an element in the array you may simply refer to it by its index and operate on it accordingly.

```
char vowels[] = {'a', 'e', 'i', 'o', 'u'};
//      indexes:  0    1    2    3    4

std::cout << vowels[0]; // Prints: a

vowels[0] = 'r';

std::cout << vowels[0]; // Prints: r
```

In the case above we initialized an array of `char`s with all of the vowels, and then printed out the first element in the array at index `0`. We then modified the element at index `0` by assigning it a new value of `r`, which got printed out below.

Arrays in C++ have a set size, meaning you can't add or remove elements once the array has been created. You may only modify existing elements without changing the total size or shape of the structure.