1 Arrays

Arrays hold a series of values, all of the same type, each of which can be assessed by a position in the array. In C++ the size of the array **must** be provided when declared. You cannot give a variable as the size — it must be a constant, or a *constant expression*.

```
int myArray[3];
myArray[0] = 0;
myArray[1] = 0;
myArray[2] = 0;
```

You can also get the same effect by using the following

```
int myArray[3] = {0};
```

Note that this is only possible if you want to initialize all values to zero. For example, the following fills only the first element in the array with the value 2 and the rest of the elements with value 0.

```
int myArray[3] = {2};
```

An array can also be initialized with an initializer list, in which case the compiler can deduce the size of the array automatically.

```
int arr[] = {1,2,3,,4}; // Compiler creates an array of 4 elements.
```

This shows a one-dimensional array, but C++ allows multi-dimensional arrays.

```
char ticTacToeBoard[3][3];
ticTacToeBoard[1][1] = 'o';
```

2 std::array

The arrays discussed in the previous section come from C, and still work in C++. However, C++ has a special type of fixed-size container called std::array, defined in the ¡array¿ header file.

There are a number of advantages in using std::arrays instead of C-style arrays.

1. They always know their own size
2. Do not automatically get cast to a pointer to avoid certain types of bugs
3. Have iterators to easily loop over the elements.

```
#include <iostream>
#include <array>
using namespace std;
int main() {
    array<int,3> arr = {9,8,7};
    cout << "Array size = " << arr.size() << endl;
    cout << "Element 2 = " << arr[1] << endl;
    return 0
}
```

Note that both C-style and std::arrays have a fixed size, which should be known at compile time. They cannot grow or shrink at run time. If you want an array with a dynamic size, it's recommended to use std::vector, explained later. A vector immediately grows in size when you add new elements to it.