**Library arrays.**

The arrays explained above are directly implemented as a language feature, inherited from the C language. They are a great feature, but by restricting its copy and easily decay into pointers, they probably suffer from an excess of optimization.

To overcome some of these issues with language built-in arrays, C++ provides an alternative array type as a standard container. It is a type template (a class template, in fact) defined in header [<array>](https://cplusplus.com/%3Carray%3E).

Containers are a library feature that falls out of the scope of this tutorial, and thus the class will not be explained in detail here. Suffice it to say that they operate in a similar way to built-in arrays, except that they allow being copied (an actually expensive operation that copies the entire block of memory, and thus to use with care) and decay into pointers only when explicitly told to do so (by means of its member data).

Just as an example, these are two versions of the same example using the language built-in array described in this chapter, and the container in the library:

|  |  |
| --- | --- |
| **language built-in array** | **container library array** |
| #include <iostream>  using namespace std;  int main()  {  int myarray[3] = {10,20,30};  for (int i=0; i<3; ++i)  ++myarray[i];  for (int elem : myarray)  cout << elem << '\n';  } | #include <iostream>  #include <array>  using namespace std;  int main()  {  array<int,3> myarray {10,20,30};  for (int i=0; i<myarray.size(); ++i)  ++myarray[i];  for (int elem : myarray)  cout << elem << '\n';  } |

As you can see, both kinds of arrays use the same syntax to access its elements: myarray[i]. Other than that, the main differences lay on the declaration of the array, and the inclusion of an additional header for the *library array*. Notice also how it is easy to access the size of the *library array*.