Arrays

Definition

An array is a continuous block of memory used for storing multiple instances of a single data type.

In C++ and most of the statically typed programming languages, the amount of space a data type takes on your RAM is fixed.

For example, an `int32_t` takes 4 bytes (32 bits) of memory. Similarly an `int64_t` takes 8 bytes of memory. So, to store 100 `int64_t`s, we already know that we'll need 800 bytes of memory.

The idea with an array then is that we just allocate a 800 byte chunk of memory that we then use to store 100 `int64_t`s. The first `int64_t` starts at the beginning of the block, the second one after that, i.e. 8 bytes after the starting position, third one is 16 bytes after so on and so forth.

In general for a data type which takes up `n` bytes of memory, if you want to store `m` instances of that data type, you need `n` * `m` bytes of memory. And the position of i^{th} instance will be `starting position + n * (i - 1)`

Since, C++ knows the size of the types and the starting position you only need to specify which element from the array you want to access. To be more precise, you have to specify the subscript of the element you want to access. Where subscript is the position of the element in the array - 1. The subscript in C++ is also often called the 0-based index of the element in the array.

Types of array

There are primarily three types of arrays:

- statically allocated arrays (C-style arrays)
- dynamically allocated static sized arrays (Java style arrays)
- dynamically allocated dynamic sized arrays (Array List / Dynamic Array / Vector)

Statically allocated arrays / C-style arrays

These arrays have a size known at compile time, and it can't be changed. This kind of array is used when you have a fixed number of variables, however that number is either too big or requires you to iterate over the variables. For example, if you wanted to count frequency of numbers entered by the user such that the number can't be lesser than 0 or greater than 100, you can use a C-style array to keep track of this.

To create a C-style array in C++ you can use one of the following two ways.

```
Type name[size] = {/* initialization values */};
// OR
#include <array>
std::array<Type, size> name{/* initialization values */};
```

The first way was inherited from C, and hence create a pure C style array. Where you just have an array like you would in C.

The second way is from the C++ standard library, and it uses `std::array` class, to create an array object.

The benefit of using `std::array` over the typical C-style array are:

- You get more utility functions.
- You get better type safety and error handling.
- You get same speed as the C array

The downside of this is that your C++ code is:

- more verbose
- not compatible with C

  [!NOTE] Depending upon what you want to do with the array, C compatibility can be essential in some scenarios, though it's not really needed most of the time.

To access an element of an array, both use the same syntax, `name[0-based index]` where `name` is the name of the array and the `0-based index` is the index of the element you want to access.

Note that if you're using C syntax (i.e. the first syntax above), you can also write `0-based index[name]` because of the way arrays used to work in C, however it's not recommended to do so because it can lead to confusing code.

Dynamically allocated static sized arrays / Java style arrays

Dynamically allocated static sized arrays are arrays that are allocated at the runtime BUT can't have their size changed.

To create a dynamically allocated static sized array in C++ you can use the following syntax.

```cpp
Type *name = new Type[size];
//Use the array
delete[] name;

// If you're using g++; the GNU C++ compiler, then you can use
// the normal array syntax without delete, however, be warned
// this is not standard and shouldn't be used without due
// diligence.
```

Note that you need to delete the array when you're done with it. If you don't, you may experience memory leaks, which are bad for, data safety.

> [!NOTE] It is almost always better to use std::vector (dynamically allocated dynamically sized array) over dynamically allocated static sized arrays in C++. This is because latter is less memory safe, has less utility functions and is more error prone.

Dynamically allocated and dynamic sized arrays / Array List / Dynamic Array / Vector

A dynamically allocated and dynamic sized array is an array that is allocated at runtime and can be resized. In C++ to create such an array, we use `std::vector`

To create a `std::vector` instance, we can use the following syntax.

```cpp
#include <vector>
std::vector<Type> name{/*initialization list*/}; // creates a vector with said elements
// or
std::vector<Type> name(size); // creates a vector of given size with garbage variables
// or
std::vector<Type> name(size, element);
// creates a vector of given size with said elements instead of garbage
```