Named Return Value Optimization

Here we learn about how compilers use NRVO and Eliding to optimize.

WE'LL COVER THE FOLLOWING

Compilers and NRVO

Compilers are even smarter, and they can elide in cases when you return a named object - it's called **Named Return Value Optimization - NRVO**

```
Test Create()
{
  Test t;
  // several instruction to initialize 't'...
  return t;
}
auto n = Create(); // temporary will be usually    elided
```

In Which Cases Is Eliding Allowed?

Currently, the Standard allows eliding in cases like:

- When a temporary object is used to initialise another object (including the object returned by a function, or the exception object created by a throw-expression)
- When a variable that is about to go out of scope is returned or thrown
- When an exception is caught by value

However, it's up to the compiler/implementation to elide or not. In practice, all the constructors' definitions are required.

> With C++17, we get clear rules on when elision has to happen, and thus constructors might be entirely omitted. In fact, instead of eliding the copies the compiler defers the “materialisation” of an object.

Now that you’re familiar with NRVO, let’s move on to an example of the non-movable/non-copyable type.