Guaranteed Copy Elision

This part introduces the concept of Copy Elision and its role in optimization.

WE'LL COVER THE FOLLOWING

- Copy Elision Example

Copy Elision is a common optimisation that avoids creating unnecessary temporary objects.

Copy Elision Example

For example:

```
#include <iostream>
using namespace std;

struct Test
{
  Test() { std::cout << "Test::Test\n"; }
  Test(const Test&) { std::cout << "Test(const Test&)\n"; }
  Test(Test&&) { std::cout << "Test(Test&&)\n"; }
  ~Test() { std::cout << "~Test\n"; }
};

Test Create()
{
  return Test();
}

int main()
{
  auto n = Create();
}
```

In the above call, you might assume a temporary copy is used - to store the return value of `Create`.

In C++14, most compilers recognise that the temporary object can be optimised easily, and they can create `n` “directly” from the call of `Create()`. So you’ll probably see the following output:

```
Test::Test // create n

~Test // destroy n when main finishes
```

In its basic form, the copy elision optimisation is called Return Value Optimisation (**RVO**).

As an experiment, in GCC you can add a compiler flag `-fno-elide-constructors` and use `-std=c++14` (or some earlier language standard). In that case you’ll see a different output:

```
// compiled as "g++ CopyElision.cpp -std=c++14 -fno-elide-constructors"

Test::Test

Test(Test&&)

~Test

Test(Test&&)

~Test

~Test
```

In this case, we have two extra copies that the compiler uses to pass the return value into `n`.

Another optimization technique in C++ 17 is NRVO. Read on to the next lesson to learn more about it.