Third standard attribute

This section delves into the details of the third standard attribute called nodiscard.

The third standard attribute we get in C++17 is:

[[nodiscard]] attribute

`[[nodiscard]]` can be applied on a function or a type declaration to mark the importance of the returned value:

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

[[nodiscard]] int Compute();

void Test()
{
  Compute(); // Warning! return value of a
          // nodiscard function is discarded
}
```

The above code should emit a warning when you compile it as you haven't assigned the result to a variable.

What it means is that you can force users to handle errors. For example, what happens if you forget about using the return value from `new` or `std::async()`?

Additionally, the attribute can be applied on types. One use case for it might be error codes:

```
enum class [[nodiscard]] ErrorCode {
    OK,
    Fatal,
    System,
    FileIssue
};
```

```
ErrorCode OpenFile(std::string_view fileName);
ErrorCode SendEmail(std::string_view sendto, std::string_view text);
ErrorCode SystemCall(std::string_view text);
```

Now, every time you'd like to call such functions, you're "forced" to check the return value. For important functions checking return codes might be crucial and using `[[nodiscard]]` might save you from a few bugs.

Using a return value

In the Standard, it's defined as "Discarded-value expressions". It means that you call a function only for its side effects. In other words, there's no if statement around or an assignment expression. In that case, when a type is marked as `[[nodiscard]]` the compiler is encouraged to report a warning.

However, to suppress the warning you can explicitly cast the return value to `void` or use `[[maybe_unused]]`:

```
[[nodiscard]] int Compute();
void Test() {
    static_cast<void>(Compute()); // fine...

    [[maybe_unused]] auto ret = Compute();
}
```

> In addition, in C++20 the Standard Library will use `[[nodiscard]]` in a few places like: `operator new`, `std::async()`, `std::allocate()`, `std::launder()`, and `std::empty()`.
> This feature was already merged into C++20 with P0600.

> The second addition to C++20 is `[[nodiscard("reason")]]`, see in P1301.

> This lets you specify why not using a returned value might generate
> issues — for example, some resource leak.

Now that you’re familiar with all three attributes. Let’s take a look at what happens when these are applied to namespaces/enumerators. Continue to the next lesson to find out!