Simplification of gl/prvalue

This section further touches upon the concept of glvalue/prvalue in C++ 17.

WE'LL COVER THE FOLLOWING

Writing pr/gl value Expressions

To support Copy Elision, the authors of the proposal provided the updated definitions of `glvalue` and `prvalue`. From the Standard:

> - `glvalue` - A `glvalue` is an expression whose evaluation computes the location of an object, bit-field, or function
> - `prvalue` - A `prvalue` is an expression whose evaluation initialises an object, bit-field, or operand of an operator, as specified by the context in which it appears

For example:

```
class X { int a; };
X{10}   // this expression is prvalue
X x;    // x is lvalue
x.a     // it's lvalue (location)
```

In short: **prvalues** perform initialisation, **glvalues** describe locations.

Let's look at gl/pr values in C++17

The C++17 Standard specifies that when there's a `prvalue` initialising some `glvalue`, then there's no need to create a temporary and we can defer its materialisation.

In C++17 Copy Elision/Deferred Temporary Materialization happens when:

- in initialisation of an object from a `prvalue`: `Type t = T()`
- in a function call where the function returns a `prvalue` - like in our examples.

There are several exceptions where the temporary is still needed:

- when a prvalue is bound to a reference
- when member access is performed on a class prvalue
- when array subscripting is performed on an array rvalue
- when an array prvalue is decayed to a pointer
- when a derived-to-base conversion is performed on a class prvalue
- when a prvalue is used as a discarded value expression

> *Extra Info:* The change was proposed in: P0135R0(reasoning) - and P0135R1(wording).

The next lesson will touch upon the topic of Dynamic Memory Allocation for Over-Aligned data in C++ 17.