Parallel Data Conversion

In this lesson, we will begin refactoring the data conversion part of our program.

WE'LL COVER THE FOLLOWING

Using `std::execution::par`

As previously discussed, we can add parallel execution to the place where we convert the data. We have lots of lines to parse, each parsing is independent.

```
[[nodiscard]] std::vector<OrderRecord>
LinesToRecords(const std::vector<std::string_view>& lines)
{
    std::vector<OrderRecord> outRecords(lines.size());
    std::transform(std::execution::par, std::begin(lines), std::end(lines),
                   std::begin(outRecords), LineToRecord);

    return outRecords;
}
```

Two things needed to change to the serial version

- we need to preallocate the vector
- we have to pass `std::execution::par` (or `par_unseq`) as the first argument

The serial code also used `std::transform`, so why cannot we just pass the execution parameter?

We can even compile it... but you should see an error like:

```
Parallel algorithms require forward iterators or stronger.
```

The reason is simple: `std::back_inserter` is very handy, but it's not a forward iterator. It inserts elements into the vector, and that causes a vector to be changed (reallocated) by multiple threads. All of the insertions would have to

be guarded by some critical section, and thus the overall performance could be weak.

Since we need to preallocate the vector, we have to consider two things:

- we pay for default construction of objects inside a vector, it's probably not a big deal when objects are relatively small, and their creation is fast.
- on the other hand, the vector is allocated once, and there is no need to grow it (copy, reallocate) as in the case of `std::back_inserter`.

Next, we'll apply parallel algorithms to the calculations in our application.