Calculations

Let's write the summing function for our CSV reader app and compile the code.

WE'LL COVER THE FOLLOWING

- Design Enhancements
- Running the Code

Once all the records are available we can compute their sum:

```
[[nodiscard]] double CalcTotalOrder(const std::vector<OrderRecord>& records,
                                    const Date& startDate, const Date& endDate)
{
    return std::accumulate(std::begin(records), std::end(records), 0.0,
        [&startDate, &endDate](double val, const OrderRecord& rec) {
            if (rec.CheckDate(startDate, endDate))
                return val + rec.CalcRecordPrice();
            else
                return val;
        }
    );
}
```

The code runs on the vector of all records and then calculates the price of each element if they fit between `startDate` and `endDate`. Then they are all summed in `std::accumulate`.

Design Enhancements

The application calculates only the sum of orders, but we could think about adding other things. For example, minimal value, maximum, average order, and other statistics.

The code uses a simple approach of loading a file into a string and then creating a temporary vector of lines. We could also enhance this by using a line iterator. It would take a large string and then return a line when you iterate.

Another idea relates to error handling. For example, rather than throwing exceptions we could enhance the conversion step by storing the number of successfully processed records.

Running the Code

The application is ready to compile, and we can run it on the example data shown in the introduction.

This should read a single file `sales/book.csv` and sum up all the records (as no dates were specified):

```
.\CalcOrdersSerial.exe .\sales\
Name Of File     | Total Orders Value
sales\book.csv   | 47.50
CalcResults: 3.13 ms
CalcTotalOrder: 0.01 ms
Parsing Strings: 0.01 ms
```

The full version of the code also includes timing measurement, so that’s why you can see that the operation took around 3ms to complete. The file handling took the most part; calculations and parsing were almost immediate.

In the next sections, you’ll see a few simple steps you can do to apply parallel algorithms.