```
// constant defining the largest year supported by our library
constexpr int k_MAX_YEAR = 2200;
```

To begin implementing `yearOfTimestamp`, it helps to start with an implementation of a solution to the reverse problem, i.e., calculating the timestamp of the start of each year, which requires an adjustment to account for leap days:

```
constexpr int numLeapYearsSinceEpoch(int year)
{
    return (year         / 4) - (year         / 100) + (year         / 400)
        - ((k_EPOCH_YEAR / 4) - (k_EPOCH_YEAR / 100) + (k_EPOCH_YEAR / 400));
}

constexpr std::time_t startOfYear(int year)
    // Return the number of seconds between the epoch and the start of the
    // specified year.  The behavior is undefined if year < k_EPOCH_YEAR or
    // year > k_MAX_YEAR.
{
    return (year - k_EPOCH_YEAR) * k_SECONDS_PER_YEAR
         + numLeapYearsSinceEpoch(year - 1) * k_SECONDS_PER_DAY;
}
```

Given these tools, we could implement `yearOfTimestamp` naively with a simple loop:

```
int yearOfTimestamp(std::time_t timestamp)
{
    int year = k_EPOCH_YEAR;
    for (; timestamp >= startOfYear(year + 1); ++year) {}
    return year;
}
```

This implementation, however, has algorithmically poor performance. While a closed-form solution to this problem is certainly possible, for expository purposes we will consider how we might, at compile time, build a lookup table of the results of `startOfYear` so that `yearOfTimestamp` can be implemented as a **binary search** on that table.

Populating a built-in array at compile time is feasible by manually writing each initializer, but a decidedly better option is to generate the sequence of numbers we want as an `std:array` where all we need is to provide the **`constexpr`** function that will take an index and produce the value we want stored at that location within the array. We will start by implementing the pieces needed to make a generic **`constexpr`** function for initializing `std::array` instances with the results of a **function object** applied to each index:

```
#include <array>   // std::array
#include <cstddef> // std::size_t

template <typename T, std::size_t N, typename F>
constexpr std::array<T, N> generateArray(const F& func);
    // Return an array arr of size N such that arr[i] == func(i) for
    // each i in the half-open range [0, N).
```