possibly noteworthy amount of runtime startup overhead. This hit at startup or on first use of the library can quickly become the next performance bottleneck that needs tackling. Initialization at startup can become increasingly problematic when linking large applications with a multitude of libraries, each of which might have moderate initialization times.

3. At this point **constexpr** comes into play as a tool to develop an option that avoids as much runtime overhead as possible. An initial such implementation puts the initialization of a **constexpr** array of values into the corresponding **inline** implementation in a library header. While this option minimizes the runtime overhead, the compile-time overhead now becomes significantly larger for every translation unit that depends on this library.

4. When faced with crippling compile times, the likely next step is to insulate the compile-time-generated table in an implementation file and to provide runtime access to it through accessor functions. While this refactoring removes the compilation overhead from clients who consume a binary distribution of the library, anyone who needs to build the library is still paying this cost each time they do a clean build. In modern environments, with widely disparate operating systems and build toolchains, source distributions have become much more common, and this overhead is imposed on a wide range of clients for a popular library.

5. Finally, the data table generation is moved into a separate program, often written in Python or some other non-C++ language. The output of this outboard program is then embedded as raw data, e.g., a sequence of numbers initializing an array, in a C++ implementation file. This solution eliminates the compile-time overhead for the C++ program; the cost of computing the table is paid only once by the developer. On the one hand, this solution adds to the maintenance costs for the initial developer, since a separate toolchain is often needed. On the other hand, the code becomes simpler, since the programmer is free to choose the best language for the job and is free from the constraints of **constexpr** in C++.

Thus, as attractive as being able to precompute values directly in compile-time C++ might seem, complex situations often dictate against that choice. Note that a programmer with this knowledge might skip all of the intermediate steps and jump straight to the last one. For example, a list of prime numbers is readily available on the Internet without needing even to write a script; a programmer need only cut and paste it once, knowing that it will *never* change.

Annoyances

Penalizing run time to enable compile time

When adopting **constexpr** functions, programmers commonly forget that these functions are also called at run time, often more frequently than at compile time. Restrictions on the