Use Cases

Nonrecursive constexpr algorithms

The C++11 restrictions on **constexpr** functions often forced programmers to implement naturally iterative algorithms in a recursive manner. Consider, as a familiar example, a naive C++11-compliant implementation of a **constexpr** function, `fib11`, returning the nth Fibonacci number:

```
constexpr long long fib11(long long x)
{
    return
        x == 0 ? 0
               : (x == 1 || x == 2) ? 1
                                    : fib11(x - 1) + fib11(x - 2);
}
```

For a more efficient (yet less intuitive) C++11 algorithm, see *Recursive Fibonacci* in *Appendix — Optimized C++11 example algorithms* on page 965.

We used **long long** (instead of **long**) here to ensure a ~~unique~~ C++ type having at least 8 bytes on all conforming platforms ~~for simplicity of exposition~~. We deliberately chose *not* to make the value returned **unsigned** because the extra bit does not justify changing the **algebra** (from **signed** to **unsigned**). For more discussion on these specific topics, see Section 1.1."**long long**" on page 89.

The implementation of the `fib11` function (above) has several undesirable properties.

1. **Reading difficulty** — Because it was implemented using a single **return** statement, branching requires a chain of *ternary operators*, leading to a single long expression that might impede human comprehension. This particular example can be written more concisely:

   ```
   constexpr long long fib11(long long x)
   {
       return x <= 1 ? x : fib11(x - 1) + fib11(x - 2);
   }
   ```

   However, not all such recurrence relations admit such simplification~~, and in any event, such cosmetic modifications have no effect on efficiency~~.

2. **Inefficiency and poor scaling** — The explosion of recursive calls is taxing on compilers: (1) the time to compile is markedly longer for the *recursive* C++11 algorithm than it would be for its *iterative* C++14 counterpart, even for modest inputs,[2] and (2)

[2] As an example, Clang 12.0.1 (c. 2021), running on an x86-64 machine, required more than 80 times longer to evaluate `fib(27)` implemented using the *recursive* (C++11) algorithm than to evaluate the same functionality implemented using the *iterative* (C++14) algorithm.

the compiler might simply refuse to complete the compile-time calculation if it exceeds some internal (platform-dependent) *threshold* for the allowed number of operations.[3]

3. **Redundancy** — Even if the recursive implementation were suitable for small input values during compile-time evaluation, it would be unlikely to be suitable for any run-time evaluation, thereby requiring programmers to provide and maintain *two* separate versions of the same algorithm: a compile-time *recursive* one and a runtime *iterative* one.

In contrast, an *imperative* implementation of a **constexpr** function returning the nth Fibonacci number in C++14, `fib14`, does not have any of the deficiencies discussed above:

```
constexpr long long fib14(long long x)
{
    if (x == 0) { return 0; }

    long long a = 0;
    long long b = 1;

    for (long long i = 2; i <= x; ++i)
    {
        long long temp = a + b;
        a = b;
        b = temp;
    }

    return b;
}
```

As one would expect, the compile time required to evaluate the iterative implementation above is manageable[4]; of course, far more computationally efficient — e.g., closed form[5] — solutions to this classic exercise are available.

[3] As an example, Clang 12.0.1 (c. 2021), running on an x86-64 machine, fails to compile `fib11(28)`:

```
error: static_assert expression is not an integral constant expression
    static_assert(fib11(28) == 317811, "");
                  ^~~~~~~~~~~~~~~~~~~

note: constexpr evaluation hit maximum step limit; possible infinite loop?
```

GCC 11.2 (c. 2021) fails at `fib(36)`, with a similar diagnostic:

```
error: 'constexpr' evaluation operation count exceeds limit of 33554432
       (use '-fconstexpr-ops-limit=' to increase the limit)
```

~~Clang 12.0.1 (c. 2021) fails to compile any attempt at constant evaluation of `fib11(28)`, with the following diagnostic message:~~

```
note: constexpr evaluation hit maximum step limit; possible infinite loop?
```

[4] Both GCC 11.2 (c. 2021) and Clang 12.0.1 (c. 2021) evaluated `fib14(46)` correctly in less than 20ms on a machine running Windows 10 x64 and equipped with an Intel Core i7-9700k CPU.

[5] E.g., see http://mathonline.wikidot.com/a-closed-form-of-the-fibonacci-sequence.