```
    static_assert(c0isdig || '\'' == c0 || '.' == c0,
                  "Invalid fixed-point digit");
    static_assert(!c0isdig || (maxData - (c0 - '0')) / 10 >= rawVal,
                  "Fixed-point overflow");

    // precision is
    // (1) < 0 if a decimal point was not seen,
    // (2) 0   if a decimal point was seen but no digits after the decimal point,
    // (3) > 0 otherwise, incremented once for each digit after the decimal point.
        static constexpr int nextPrecision = ('\'' == c0    ? precision :
                                              '.'  == c0    ?         0 :
                                              precision < 0 ?        -1 :
                                              precision + 1);

    // Instantiate this template recursively to consume remaining characters.
    using RecurseType = MakeFixedPoint<(c0isdig ? 10 * rawVal + c0 - '0' :
                                        rawVal), nextPrecision, c...>;

public:
    using type = typename RecurseType::type;
    static constexpr type makeValue() { return RecurseType::makeValue(); }
        // Return the computed fixed-point number.
};
```

This specialization consumes one character, `c0`, from the parameter pack. The first **`static_assert`** checks that `c0` is either a digit, digit separator (`'\''`), or decimal point (`'.'`). The second **`static_assert`** checks that the computation is not in danger of overflowing the maximum value of a **`long long`**. The constant, `nextPrecision`, which will be passed to the recursive instantiation of this template, keeps track of how many digits have been consumed after the decimal point (or `-1` if the decimal point has not yet been consumed). The `RecurseType` alias is the recursive instantiation of this template with the updated raw value (after consuming `c0`), the updated precision, and the input character sequence after having dropped `c0`. Thus, each recursion gets a potentially larger `rawVal`, a potentially larger `precision`, and a shorter list of unconsumed characters. The definitions of `type` and `makeValue` simply defer to the definitions in the recursive instantiation.

Finally, we define the `_fixed` UDL operator template, instantiating `MakeFixedPoint` with an initial `rawVal` of `0`, an initial `precision` of `-1`, and with a `c...` parameter pack consisting of all of the characters in the naked literal:

```
template <char... c>
constexpr typename MakeFixedPoint<0, -1, c...>::type operator""_fixed()
{
    return MakeFixedPoint<0, -1, c...>::makeValue();
}

}  // Close namespace literals
}  // Close namespace fixedpoint
```