The delimiter can be — and, in practice, often is — an empty character sequence:

```
const char s7[] = R"("Hello, World!")";
    // OK, equivalent to "\"Hello, World!\""
```

A nonempty delimiter — e.g., `!` — can be used to disambiguate any appearance of the `)"` character sequence within the literal data:

```
const char s8[] = R"!("--- R"(Raw literals are not recursive!)" ---")!";
    // OK, equivalent to "\"--- R\"(Raw literals are not recursive!)\" ---\""
```

Had an empty delimiter been used to initialize `s8` in the example above, the compiler would have produced a perhaps obscure compile-time error:

```
const char s8a[] = R"("---R"( Raw literals are not recursive!)" ---")";
    //                                                        ^~
    // Error, decrement of read-only location
```

In fact, it could turn out that a program with an unexpectedly terminated *raw* string literal could still be valid and compile quietly:

```
void emitPith()
{
    printf(R"("Live-Free, don't (ever)","Die!");
        // prints "Live-Free, don't (ever

    printf((R"("Live-Free, don't (ever)","Die!"));
        // prints Die!
}
```

Fortunately, examples like the one above are invariably contrived, not accidental.

Use Cases

Embedding code in a C++ program

When a source code snippet needs to be embedded as part of the source code of a C++ program, use of a *raw* string literal can significantly reduce the syntactic noise that would otherwise be caused by repeated escape sequences. As an example, consider a regular expression for an online shopping product ID represented as a conventional string literal:

```
const char* productIdRegex = "[0-9]{5}\\(\".*\"\\)";
    // This regular expression matches strings like 12345("Product").
```

Not only do the backslashes obscure the meaning to human readers, a mechanical translation is often needed when transforming between source and data, such as when copying the contents of the string literal into an online regular-expression validation tool, and introduces significant opportunities for human error. Using a raw string literal solves these problems:

```
const char* productIdRegex = R"([0-9]{5}\(".*"\))";
```