If we use the basic syntax for a *raw* string literal, we will get a syntax error:

```
const char s3[] = R"(printf("printf(\"Hello, World!\")"))";  // collision
//                                                    ^^
//                      syntax error after literal ends
```

To circumvent this problem, we could escape every special character in the string separately, as in C++03, but the result is difficult to read and error prone:

```
const char s4[] = "printf(\"printf(\\\"Hello, World!\\\")\")";  // error prone
```

Instead, we can use the extended disambiguation syntax of *raw* string literals to resolve the issue:

```
const char s5[] = R"###(printf("printf(\"Hello, World!\")"))###";  // cleaner
```

This disambiguation syntax allows us to insert an essentially arbitrary sequence of characters between the outermost quote/parenthesis pairs such that the combined sequence — e.g., `)###"` — avoids the collision with the literal data:

```
//                      delimiter and parenthesis
//                   v~~~                          ~~~v
const char s6[] = R"xyz(<-- Literal String Data -->)xyz";
//                 ^     ^~~~~~~~~~~~~~~~~~~~~~~~~~^
//                 |           string contents
//                 |
//                 | uppercase R
```

The delimiter of a raw string literal can comprise any member of the **basic source character set** except space, backslash, parentheses, and the control characters representing horizontal tab, vertical tab, form feed, and new line.

The value of `s6` above is equivalent to `"<-- Literal String Data -->"`. Every raw string literal comprises these syntactical elements in order:

- An uppercase `R`
- The opening double quotes, `"`
- An optional ~~arbitrary~~ sequence of characters called the *delimiter* (e.g., `xyz`)
- An opening parenthesis, `(`
- The contents of the string
- A closing parenthesis, `)`
- The same delimiter specified previously, if any (i.e., `xyz`, not reversed)
- The closing double quotes, `"`