```
auto printTypeInformation = [](auto object)
{
    std::cout << "     size: " << sizeof(object) << '\n'
              << "alignment: " << alignof(decltype(object)) << '\n';
};
```

Because there is no explicit type available within the body of the `printTypeInformation` lambda,[7] a programmer aiming to remain entirely within the C++ Standard[8] is forced to use the **`decltype`** construct explicitly to first obtain the type of `object` before passing it on to **`alignof`**.

See Also

- "**`decltype`**" (§1.1, p. 25) explains how **`decltype`** helps work around **`alignof`**'s limitation of accepting only a type, not an expression (see *Annoyances — **`alignof`** is defined only on types* on page 193).
- "**`alignas`**" (§2.1, p. 168) discusses how **`alignas`** can be used to provide an artificially stricter alignment, e.g., more than natural alignment.

[7]In C++20, referring to the type of a generic lambda parameter explicitly is possible, due to the addition to lambdas of some familiar template syntax:

```
auto printTypeInformation = []<typename T>(T object)
{
    std::cout << "     size: " << sizeof(T) << '\n'
              << "alignment: " << alignof(T) << '\n';
};
```

[8]Note that **`alignof`**`(object)` will work on every major compiler — GCC 11.2 (c. 2021), Clang 12.0.1 (c. 2021), and MSVC 19.29 (c. 2021) — as a nonstandard extension.