# The `alignof` Operator

The keyword **`alignof`** serves as a compile-time operator used to query the **alignment requirements** of a type on the current platform.

## Description

The **`alignof`** operator, when applied to a type, evaluates to an **integral constant expression** that represents the alignment requirement of its argument type. Similar to **`sizeof`**, the compile-time value of **`alignof`** is of type `std::size_t`; unlike **`sizeof`** that can accept an arbitrary expression, **`alignof`** is defined for only type identifiers but often works on expressions anyway (see *Annoyances* — ***`alignof` is defined only on types*** on page 193). The argument type, `T`, supplied to **`alignof`** must be a **complete type**, ~~a reference type, or an array type~~. If `T` is a complete type, the result is the alignment requirement for `T`. If `T` is a reference type, the result is the alignment requirement for the referenced type. If `T` is an array type, the result is the alignment requirement for every element in the array. For example, on a platform where **`sizeof(short)`** `== 2` and **`alignof(short)`** `== 2`, the following assertions pass:

```
static_assert(alignof(short)    == 2, "");  // complete type   (sizeof is 2)
static_assert(alignof(short&)   == 2, "");  // reference type  (sizeof is 2)
static_assert(alignof(short[5]) == 2, "");  // array type      (sizeof is 10)
static_assert(alignof(short[])  == 2, "");  // array type      (sizeof fails)
```

According to the C++11 Standard, "An object of array type contains a contiguously allocated nonempty set of `N` subobjects of type `T`."[1] Note that, for every type `T`, **`sizeof(T)`** is always a multiple of **`alignof(T)`**; otherwise, storing multiple `T` instances in an array would be impossible without padding, and the Standard explicitly prohibits padding between array elements.

### `alignof` Fundamental Types

Like their size, the alignment requirements of a **`char`**, **`signed char`**, and **`unsigned char`** are guaranteed to be 1 on every conforming platform. For any other fundamental or pointer type `FPT`, **`alignof(FPT)`** is platform-dependent but is typically approximated well by the type's **natural alignment** — i.e., **`sizeof(FPT)`** `==` **`alignof(FPT)`**:

```
static_assert(alignof(char)   == 1, "");  // guaranteed to be 1
static_assert(alignof(short)  == 2, "");  // platform-dependent
static_assert(alignof(int)    == 4, "");  //      "        "
static_assert(alignof(double) == 8, "");  //      "        "
static_assert(alignof(void*)  >= 4, "");  //      "        "
```

[1] **iso11a**, section 8.3.4, "Arrays," paragraph 1, p. 188