

---

## The alignas Specifier

The keyword **alignas** can be used in the **declaration** of a **class type**, a **data member**, an enumeration, or a **variable** to strengthen its **alignment**.

### Description

Each object type in C++ has an **alignment requirement** that restricts the addresses at which an object of that type is permitted to reside within the virtual-memory-**address space**. The **alignment requirement** is imposed by the object type on all objects of that type. The **alignas** specifier provides a means of specifying a stricter **alignment requirement** than dictated by the type itself for a particular **variable** of the type or an individual **data member** of a **user-defined type (UDT)**. The **alignas** specifier can also be applied to a UDT itself, but see *Potential Pitfalls — Applying alignas to a type might be misleading* on page 177.

### Supported alignments

An **alignment value** is an integral of type `std::size_t` that represents the number of **bytes** between the addresses at which a given object may be allocated. In practice, the **alignment value** will always evenly divide the numerical value of the address of any object of that type. All **alignment values** in C++ are non-negative powers of two and are divided into two categories depending on whether they are larger than the **alignment requirement** of the `std::max_align_t` type. The `std::max_align_t` type's **alignment requirement** is at least as strict as that of every **scalar type**. An **alignment value** of less than or equal to the **alignment requirement** of `std::max_align_t` is a **fundamental alignment**; otherwise, it is an **extended alignment**. The `std::max_align_t` type is typically an alias to the largest **scalar type**, which is **long double** on most platforms, and its **alignment requirement** is usually 8 or 16.

**Fundamental alignments** are required to be supported in *all* contexts, i.e., for variables with automatic, static, and dynamic storage durations as well as for **nonstatic data members** of a class and for function **arguments**. While all fundamental and pointer types have **fundamental alignments**, their specific **values** are **implementation defined** and might differ between platforms. For example, the **alignment requirement** of type **long** might be 4 on MSVC and 8 on GCC.