In contrast, whether *any* extended alignment is supported at all and, if so, in which contexts, is implementation defined.[1] For example, the strictest supported extended alignment for a variable with static storage duration might be as large as 2^{28} or 2^{29} or as small as 2^{13}.

Since many aspects pertaining to the alignment requirements are implementation defined, we will use a specific platform to illustrate the behavior of **alignas** throughout this feature's section. Accordingly, the examples below show the behavior observed for the Clang compiler targeting desktop x86-64 Linux.

Strengthening the alignment of a particular object

In its most basic form, the **alignas** specifier strengthens the alignment requirement of a particular object. The desired alignment requirement is an **integral constant expression** provided as an argument to **alignas**:

```
alignas(8) int i;     // OK, i is aligned on an 8-byte address boundary.
int j alignas(8), k;  // OK, j is 8 byte aligned; alignment of k is unchanged.
```

If more than one alignment pertains to a given object, the strictest alignment value is applied:

```
alignas(4) alignas(8) alignas(2) char m;  // OK, m is 8-byte aligned.
alignas(8) int n alignas(16);             // OK, n is 16-byte aligned.
```

For a program to be well formed, a specified alignment value must satisfy three requirements.

1. Be either zero or a non-negative integral power of two of type `std::size_t` (0, 1, 2, 4, 8, 16…)
2. Be larger or equal to what the alignment requirement would be without the **alignas** specifier
3. Be supported on the platform in the context in which the entity appears

[1] Implementations might warn when the alignment of a global object exceeds some maximal hardware threshold, such as the size of a physical memory page, e.g., 4096 or 8192. For automatic variables defined on the program stack, making alignment more restrictive than what would naturally be employed is seldom desired because at most one thread is able to access proximately located variables there unless explicitly passed in via address to separate threads; see *Use Cases — Avoiding false sharing among distinct objects in a multithreaded program* on page 174. Note that, in the case of `i0` in the **alignas**(32) line in the first code snippet on page 170, a conforming platform that did not support an extended alignment of 32 would be required to report an error at compile time.