Additionally, if the specified alignment value is zero, the **alignas** specifier is ignored:

```
// static variables declared at namespace scope
alignas(32) int i0; // OK, 32-byte aligned (extended alignment)
alignas(16) int i1; // OK, 16-byte aligned (strictest fundamental alignment)
alignas(8)  int i2; // OK,  8-byte aligned (fundamental alignment)
alignas(7)  int i3; // Error, not a power of two
alignas(4)  int i4; // OK, no change to alignment requirement
alignas(2)  int i5; // Error, less than alignment would be without alignas
alignas(0)  int i6; // OK, alignas specifier ignored

alignas(1024 * 16) int i7;
    // OK, might warn on other platforms, e.g., exceeds physical page size

alignas(1 << 30) int i8;
    // Error, exceeds maximum supported extended alignment

alignas(8) char buf[128]; // OK, 8-byte-aligned, 128-byte character buffer

void f()
{
  // automatic variables declared at function scope
  alignas(4)  double e0; // Error, less than alignment would be without alignas
  alignas(8)  double e1; // OK, no change to 8-byte alignment requirement
  alignas(16) double e2; // OK, 16-byte aligned (fundamental alignment)
  alignas(32) double e3; // OK, 32-byte aligned (extended alignment)
}
```

Strengthening the alignment of individual data members

Within a user-defined type (**class**, **struct**, or **union**), using the **alignas** keyword to specify the alignments of individual data members is possible:

```
struct T2
{
    alignas(8)  char   x;  // size 1; alignment 8
    alignas(16) int    y;  // size 4; alignment 16
    alignas(64) double z;  // size 8; alignment 64
};  // size 128; alignment 64
```

The effect here is the same as if we had added the padding explicitly and then set the alignment of the structure overall:

```
struct alignas(64) T3
{
    char   x;      // size 1; alignment 8
    char   a[15];  // padding
```