```
struct Foo;                 // OK, does not specify an alignment
struct alignas(double) Foo; // OK, must be equivalent to every definition
struct alignas(8) Foo;      // OK, all definitions must be identical.

struct alignas(8) Foo { };  // OK, def. equiv. to each decl. specifying alignment

struct Foo;                 // OK, has no effect
struct alignas(8) Foo;      // OK, has no effect; might warn after definition
```

Specifying an alignment in a forward declaration without specifying an equivalent one in the defining declaration is ill formed, no diagnostic required (IFNDR) if the two declarations appear in distinct translation units:

```
struct alignas(4) Bar;      // OK, forward declaration
struct Bar { };             // Error, missing alignas specifier

struct alignas(4) Baz;      // OK, forward declaration
struct alignas(8) Baz { };  // Error, nonequivalent alignas specifier
```

Both of the errors above are flagged by Clang. MSVC and ICC warn on the first one and produce an error on the second one, whereas neither of them is reported by GCC. Note that when the inconsistency *occurs across translation units*, no mainstream compiler is likely to diagnose the problem:

```
// file1.cpp:
struct Bam { char ch; } bam, *p = &bam;

// file2.cpp:
struct alignas(int) Bam;  // Error, definition of Bam lacks alignment specifier.
extern Bam* p;            //        (no diagnostic required)
```

Any program incorporating both translation units above is IFNDR.

Applying `alignas` to a *type* might be misleading

When applying the **`alignas`** specifier to a user-defined type having no base classes, one might be convinced that it is equivalent to applying **`alignas`** to its first declared data member:

```
struct S0 {
    alignas(16) char d_buffer[128];  // guaranteed to be 16-byte aligned
                int  d_index;
};

struct alignas(16) S1 {
    char d_buffer[128];              // guaranteed to be 16-byte aligned
    int  d_index;
};
```