every POD type in C++ has a corresponding type in C that — at least in practice — is structurally compatible with it, even if its C rendering lacks certain member functions, access controls, empty base classes, and so on that might otherwise pertain in C++; see *Use Cases — Translating a C++-only type to C (standard layout)* on page 452.

Being a C++ POD-**struct**, though almost always an overly strict constraint, is sufficient to guarantee many other useful properties not generally afforded to other **class types**. The details of the minimal requirements needed for any given property to hold are discussed in subsequent sections; e.g., see *Standard-layout types* on page 417 and *Trivial types* on page 425. Let's now consider some of the special properties and advantages that *all* PODs enjoy.

1. **Contiguous storage** — All objects of POD type, a.k.a. POD objects, occupy contiguous bytes of storage. The **value representation** of a POD object is a subset of the bits in that storage, and the valid **values** of a POD object are an **implementation-defined** set of values that those bits can take on. Consider a POD-**struct**, `S1`, containing a **char** and a **short**:

   ```
   struct S1     // POD-struct whose size is typically 4 bytes
   {
       char  a;     // always exactly 1 byte
                    // typically 1 byte of padding for alignment purposes
       short b;     // at least (and typically exactly) 2 bytes
   };
   ```

   Objects of this POD type are typically stored in exactly 4 contiguous bytes and have a value representation of 24 noncontiguous bits. The 8 extra padding bits are not part of the value representation.

   Objects with virtual base classes might potentially have not just a noncontiguous value representation but also a noncontiguous **object representation**, since the virtual base subobject might not be adjacent to the rest of the object. This reason is one of a few that explains why types with virtual base classes are not POD types.

2. **Predictable layout** — The layout of every POD object is stable and in some important ways predictable. For example, the first non**static** data member (e.g., `x`) of a POD-**struct** (e.g., `X`) is guaranteed to reside at the same address as does the POD-**struct** object (e.g., `pso`) itself:

   ```
   struct X { int x; } pso;  // POD-struct object
   static_assert(static_cast<void*>(&pso) == static_cast<void*>(&pso.x), "");
   ```

   This property of a POD-**struct** predates C++03 and is true even in C. Although base classes are not permitted for C++03 POD types, in C++11 the address of a POD-**struct** having one or more base classes is the same as that of its first base class: