has been initialized. For example, consider a simple **standard-layout class**, `SS`, containing three public data members, of types **`int`**, **`double`**, and **`void`**`*`, along with some other similar standard-layout class types, `S0`, `S1`, `S2`, and `S3`, that are in turn comprised into a union, `U`:

```
struct SS { int  i; double d; void* p; };

struct S0 { ~~long~~ i; double d; void* p; };        // 0 member CIMS
struct S1 { int  j; float  d; void* p; };        // 1    "     " w/ all but S0
struct S2 { int  j; double e; char* p; };        // 2    "     " with SS, S3
struct S3 { int  j; double e; void* q; S0 s; };  // 3    "     " with SS

union U { SS ss; S0 s0; S1 s1; S2 s2; S3 s3; };  // all standard-layout types
```

In the example above, the type of the first data member of `S0` differs from that of `SS` and therefore shares no CIMS with `SS` or any of the other members of `U`. The first data member of `S1` matches exactly that of `SS` (and all of the other members of `U` except `S0`) but differs in the type of its second member; hence, `SS` and `S1` share a CIMS of length 1: **`int`**. The first two data members of `S2` exactly match those of `SS` (but differ after that), so they share a CIMS of length 2: **`int`**, **`double`**. Finally, the first three data members of `S3` exactly match those of `SS`, so they share a CIMS of length 3: **`int`**, **`double`**, **`void`**`*`.

If we create an instance of our **`union`** `U` (e.g., `u`) with `ss` as the active member and initialize the three data members of `SS`, we are able to safely access none, some, or all of those values via the other members of `U` depending on the length of their mutual CIMS:

```
U u = { 3, 5.5, 0 };  // braced initialization of SS standard-layout member

int    i0 = u.s0.i;  // Bug, no CIMS with SS

int    i1 = u.s1.j;  // OK, j member of S1 is part of CIMS with SS.
double d1 = u.s1.d;  // Bug, d member of S1 is not part of CIMS with SS.
void*  p1 = u.s1.p;  // Bug, p    "    "  "  "  "    "   "   "    "   "

int    i2 = u.s2.j;  // OK, j member of S2 is part of CIMS with SS.
double d2 = u.s2.e;  // OK, e    "    "  "  "   "   "   "    "   "
void*  p2 = u.s2.p;  // Bug, p member of S2 is not part of CIMS with SS.

int    i3 = u.s3.j;  // OK, j member of S3 is part of CIMS with SS.
double d3 = u.s3.e;  // OK, e    "    "  "  "   "   "   "    "   "
void*  p3 = u.s3.q;  // OK, q    "    "  "  "   "   "   "    "   "
```

According to the definition of standard-layout class types (see *Standard-layout types* on page 417, above), at most one class in any class hierarchy is permitted to contain non**`static`** member data; hence, the CIMS is independent of where in an inheritance hierarchy the CIMS is defined: